CHAPTER 4

Local Feature Design Concepts, Classification, and Learning

> *"Science, my boy, is made up of mistakes, but they are mistakes which it is useful to make, because they lead little by little to the truth."*
>
> — *Jules Verne,* Journey to The Center of The Earth

In this chapter we examine several concepts related to local feature descriptor design—namely local patterns, shapes, spectra, distance functions, classification, matching, and object recognition. The main focus is *local feature metrics,* as shown in Figure 4-1. This discussion follows the general vision taxonomy that will be presented in Chapter 5, and includes key fundamentals for understanding interest point detectors and feature descriptors, as will be surveyed in Chapter 6, including selected concepts common to both detector and descriptor methods. Note that the opportunity always exists to modify as well as mix and match detectors and descriptors to achieve the best results.

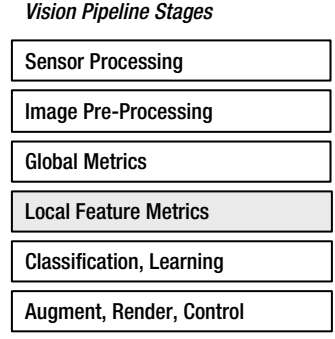


Figure 4-1. *Various stages in the vision pipeline; this chapter will focus on local feature metrics and classification and learning*

Local Features

We focus on the design of *local feature descriptors* and how they are used in training, classification, and machine learning. The discussion follows the feature taxonomy as is presented in Chapter 5 and as is illustrated in Figure 5-1. The main elements are: (1) *shape* (for example, rectangle or circle); (2) *pattern* (either dense sampling or sparse sampling); and (3) *spectra* (binary values, scalars, sparse codes, or other values). A dense patterned feature will use each pixel in the local region, such as each pixel in a rectangle, while a sparse feature will use only selected pixels from the region.

In addition to the many approaches to shape and pattern, there are numerous approaches taken for the spectra, ranging from gradient-based patch methods to sparse local binary pattern methods. The main topics covered here include:

- **Detectors**, used to locate interesting features in the image.
- **Descriptors**, used to describe the regions surrounding interesting features.
- **Descriptor attributes**, such as feature robustness and invariance.
- **Classification**, used to create databases of features and optimal feature matching.
- **Recognition**, used to match detected features in target images against trained features.
- **Feature learning**, or machine learning methods.

Based on the concepts presented this chapter, the vision taxonomy offered in Chapter 5 provides a way to summarize and analyze the feature descriptors and their attributes, thereby enabling limited comparison between the different approaches.

Detectors, Interest Points, Keypoints, Anchor Points, Landmarks

A *detector* finds interesting features in the image. The terminology in the literature for discussing an "interesting feature" includes several interchangeable terms, such as *keypoint*, *landmark, interest point,* or *anchor point*, all of which refer to features such as corners, edges, or patterns that can be found repeatedly with high likelihood. In Chapter 6, we will survey many detector methods, along with various design approaches. In some cases, the keypoint detector is used to determine the orientation vector of the surrounding feature descriptor—for example, by computing the overall gradient orientation of the corner. The uncertain or low-quality keypoints are commonly filtered out prior to feature description. Note that many keypoint methods operate on smaller pixel regions, such as 3x3 for the LBP and 7x7 for FAST.

The keypoint location itself may not be enough for feature matching; however, some methods discussed here rely on *keypoints only,* without a feature descriptor. Feature description provides more information around each keypoint, and may be computed over larger regions and multiple scales, such as SIFT and ORB.

Descriptors, Feature Description, Feature Extraction

A feature *descriptor* can be computed at each key point to provide more information about the pixel region surrounding the keypoint. However, in methods that compute features across a fixed-size pixel grid such as the Viola Jones method [146], no interest point is necessary, since the grid defines the descriptor region. Feature description typically uses some combination of color or gray scale intensity channels, as well as local information such as gradients and colors. Feature description takes place over a shape, such as a square or circle. In some cases, pixel point-pair sample patterns are used to compute or compare selected pixel values to yield a *descriptor vector*—for example, as shown later, in Figure 4-8.

Typically, an interest point provides some amount of invariance and robustness—for example, in scale and rotation. In many cases, the orientation of the descriptor is determined from the interest point, and the descriptor provides other invariance attributes. Combining the interest point with the descriptor provides a larger set of invariance attributes. And if several descriptors are associated together from the same object, object recognition is possible.

For example, a descriptor may contain multivariate, multidimensional, and multigeometric quantities calculated over several intensity channels, multiple geometric scales, and multiple perspectives. A *multivariate* descriptor may contain RGBD data (red, green, blue, and Z depth data); a *multidimensional* descriptor may contain feature descriptions at various levels of zoom across an image pyramid; and a *multigeometry* descriptor may contain a set of feature descriptions computed across affine transforms of the local image patch or region.

There is no right or wrong method for designing features; many approaches are taken. For example, a set of metrics including region shape, region texture, and region color of an object may be helpful in an application to locate fruit, while another application may not need color or shape and can rely instead on sets of interest points, feature descriptors, and their spatial relationships. In fact, combining several weaker descriptor methods into a multivariate descriptor is often the best approach.

Computing feature descriptors from an image is commonly referred to as *feature extraction*.

Sparse Local Pattern Methods

While some methods describe features densely within regular sampling grids across an image, such as the PHOG [191] method discussed in Chapter 6, other methods such as FREAK [130] use *sparse local patterns* to sample pixels anchored at interest points to create the descriptor. Depending on the method, the shapes may be trained, learned, or chosen by design, and many topologies of shapes and patterns are in current use.

To frame the discussion on sparse local pattern and descriptor methods, notice that there is a contrast with global and regional descriptor methods, which typically do *not* rely on sparse local patterns. Instead, global and regional methods typically use dense sampling of larger shapes such as rectangles or other polygons. For example, polygon shape descriptors, as will be discussed in Chapter 6, may delineate or segment the feature region using dense methods such as mathematical morphology and region segmentation. Global and regional descriptor metrics, such as texture metrics, histograms, or SDMs discussed in Chapter 3, are typically computed across cohesive, dense regions rather than sparse regions.

Local Feature Attributes

This section discusses how features are chosen to provide the desired attributes of feature goodness, such as invariance and robustness.

Choosing Feature Descriptors and Interest Points

Both the interest point detector and the feature description method must be chosen to work well together, and to work well for the type of images being processed. Robustness attributes such as contrast, scale, and rotation must be considered for both the detector and the descriptor pair. As shown in Appendix A, each interest point detector is best designed to find specific types of features, and therefore no single method is good for all types of images.

For example, FAST and Harris methods typically find many small *mono-scale* interest points, while other methods, such as that used in SIFT find fewer, larger and finely tuned *multi-scale* interest points. Some tuning of the interest point detector parameters is expected, so as to make them work at all, or else some pre-processing of the images maybe needed to help the detector find the interest points in the first place. (Chapter 6 provides a survey of interest point methods and background mathematical concepts.)

Feature Descriptors and Feature Matching

Feature description is foundational to *feature matching*, leading to image understanding, scene analysis, and object tracking. The central problems in feature matching include how to determine if a feature is differentiated from other similar features, and if the feature is part of a larger object.

The method of determining a feature match is critical, for many reasons; these reasons include compute cost, memory size, repeatability, accuracy, and robustness. While a perfect match is ideal, in practice a relative match is determined by a *distance function,* where the incoming set of feature descriptors is compared with known feature descriptors. But we'll discuss several distance functions later in this chapter.

Criteria for Goodness

Measuring the goodness of features can be done *one attribute at a time.* A general list of goodness attributes for feature landmarks is provided later, in Table 4-2. Note that this list is primarily about invariance and robustness: these are the key concepts, since performance can be tuned using various optimization methods, as will be discussed in Chapter 8. Of course, in a given application some attributes of goodness are more important than others; this is discussed in Chapter 7, in connection with ground truth data.

How do we know a feature is *good* for an application? We may divide the discussion between the interest point methods and the descriptor method, and the combined robustness and invariance attributes provided by both (see Table 4-1). The interest point at which the feature is anchored is critical, since if the anchor is not good and cannot be easily and repeatedly found, the resulting descriptor is calculated at a suboptimal location.

Table 4-1. *Some Attributes for Good Feature Descriptors and Interest Points. (See also Figure 5-2 for the general robustness criteria)*

Good Feature Metric Attributes	Details
Scale invariance	Should be able to find the feature at different scales
Perspective invariance	Should be able to find the feature from different perspectives in the field of view
Rotational invariance	The feature should be recognized in various rotations within the image plane
Translation invariance	The feature should be recognized in various positions in the FOV
Reflection invariance	The feature should be recognized as a mirror image of itself
Affine invariance	The feature should be recognized under affine transforms
Noise invariance	The feature should be detectable in the presence of noise
Illumination invariance	The feature should be recognizable in various lighting conditions including changes in brightness and contrast
Compute efficiency	The feature descriptor should be efficient to compute and match
Distinctiveness	The feature should be distinct and detectable, with a low probability of mis-match, amenable to matching from a database of features
Compact to describe	The feature should not require large amounts of memory to hold details
Occlusion robustness	The feature or set of features can be described and detected when parts of the feature or feature set are occluded
Focus or blur robustness	The feature or set of features can be detected at varying degrees of focus (i.e, image pyramids can provide some of this capability)
Clutter and outlier robustness	The feature or set of features can be detected in the presence of outlier features and clutter

Note that in many cases, image pre-processing is key to creating a good feature (Figure 4-1). If the image data has problems that can be corrected or improved, the feature description should be done after the image pre-processing. Note that many feature description methods rely on local image enhancements during descriptor creation, such as Gaussian blur of regions around keypoints for noise removal, so image pre-processing should complement the descriptor method. Each pre-processing method has drawbacks and advantages; see Table 2-1 and Chapter 2 for information on image pre-preprocessing.

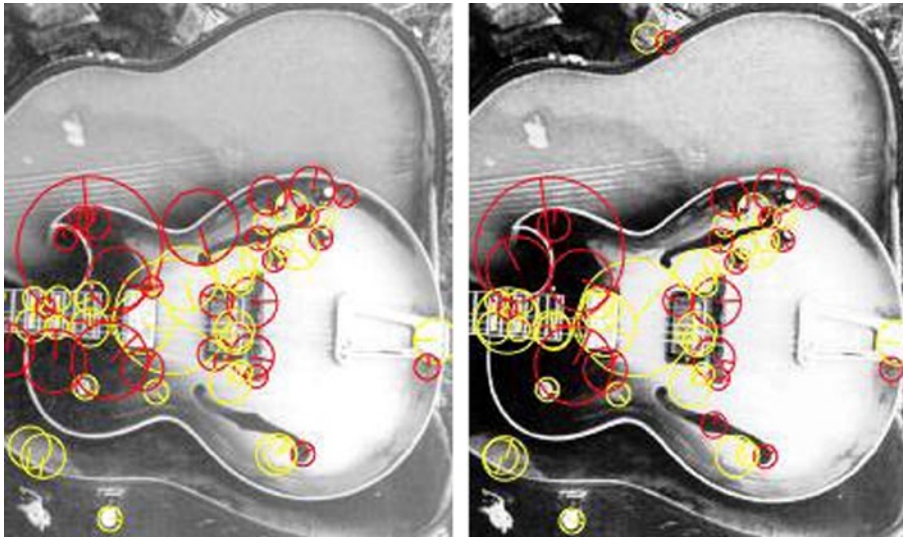


Figure 4-2. *(Left) SURF feature descriptors calculated over original image. (Right) Image has been pre-processed using histogram equalization prior to feature extraction and therefore a different but overlapping set of features is found*

Repeatability, Easy vs. Hard to Find

Ideally the feature will be easy to find in an image, meaning that the feature description contains sufficient information to be robust under various conditions (as shown in Table 4-1), such as contrast and brightness variations, scale, and rotation. *Repeatability* applies particularly to interest point detection, so the choice of interest point detector method is critical. (Appendix A contains example images showing interesting nonrepeatability anomalies for several common interest point detectors.)

Some descriptors, such as SIFT [161,178], are known to be robust under many imaging conditions. This is not too surprising, since SIFT is designed to be discriminating over multiple dimensions, such as scale and rotation, using carefully composed sets of local region gradients with a weighting factor applied to increase the importance of gradients closer to the center of the feature. But the robustness and repeatability come at a compute cost. SIFT [161,178] is one of the most computationally expensive methods; however, Chapter 6 surveys various SIFT optimizations and variations.

Distinctive vs. Indistinctive

A descriptor is distinctive if:

- The feature can be differentiated from other, similar feature regions of the image.
- Different feature vectors are unique in the feature set.
- The feature can be matched effectively using a suitable distance function.

A feature is indistinct if similar features cannot be distinguished; this may be caused by a lack of suitable image pre-processing, insufficient information in the descriptor, or an unsuitable distance function chosen for the matching stage. Of course, adding information into a simpler descriptor to make the descriptor a hybrid multivariate or multi-scale descriptor may be all that is needed to improve distinctiveness. For example, color information can be added to distinguish between skin tones.

Relative and Absolute Position

Positional information, such as coordinates, can be critical for feature goodness. For example, to associate features together using constraints on the corner position of human eyes, interest point coordinates are needed. These enable more accurate identification and location of the eyes by using, as part of an intelligent matching process, the distance and angles between the eye corner locations.

With the increasing use of depth sensors, simply providing the Z or depth location of the feature in the descriptor itself may be enough to easily distinguish a feature from the background. Position in the depth field is a powerful bit of information, and since computer vision is often concerned with finding 3D information in a 2D image field, the Z depth information can be an invaluable attribute for feature goodness.

Matching Cost and Correspondence

Feature matching is a measurement of the correspondence between two or more features using a *distance function (*discussed next in this section). Note here that feature matching has a cost in terms of memory and compute time. For example, if a feature descriptor is composed of an array of 8-bit bytes, such as an 18x18 pixel correlation template, then the feature matching cost is the compute time and memory required to compare two 18x18 (324) pixel regions against each other, where the matching method or distance function used may be SAD, SSD, or similar difference metric. Another example involves local binary descriptors such as the LBP (linear binary patterns), which are stored as bit vectors, where the matching cost is the time to perform the Hamming distance function, which operates by comparing two binary vectors via Boolean XOR followed by a bit count to provide the match metric.

In general, distance functions are well-known mathematical functions that are applied to computer vision; however, some are better suited than others in terms of computability and application to a specific vision task. For example, SSD, SAD, cosine

distance, and Hamming distance metrics have been implemented in silicon as computer machine language instructions in some architectures, owing to their wide applicability. So choosing a distance function that is accelerated in silicon can be an advantage.

The feature database is another component of the matching cost, so the organization of the database and feature search contribute to the cost. We briefly touch on this topic later in this chapter.

Distance Functions

This section provides a general discussion of distance functions used for clustering, classification, and feature matching. Note that distance functions can be taken over several dimensions—for example, 2D image arrays for feature descriptor matching, 3D voxel volumes for point cloud matching, and multidimensional spaces for multivariate descriptors. Since this is a brief overview, a deeper treatment is available by Pele[548].

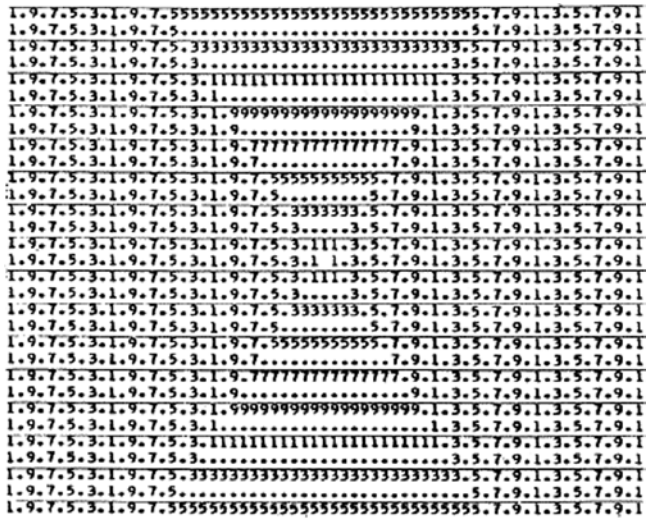
Note that kernel machines [361,362], discussed later in this chapter, provide an automated framework to classify a feature space and substitute chosen distance function kernels.

Early Work on Distance Functions

In 1968, Rosenfeld and Pfaltz[121] developed novel methods for determining the distance between image features, which they referred to as "a given subset of the picture," where the feature shapes used in their work included diamonds, squares, and triangles. The distance metrics they studied include some methods that are no longer in common use today:

- Hexagonal distance from a single point (Cartesian array)
- Hexagonal distance from a single point (staggered array)
- Octagonal distance from a single point
- City block distance from blank areas
- Square distances from blank areas
- Hexagonal distance from blank areas
- Octagonal distance from blank areas
- Nearest integer to Euclidean distance from a single point

This early work by Rosenfeld and Pfaltz is fascinating, since the output device used to render the images was ASCII characters printed on a CRT terminal or line printer, as shown in Figure 4-3.



"Square" distances (d_2) from a single point.

Figure 4-3. *An early Rosenfeld and Pfaltz rendering that illustrates a distance function (square distance in this case) using a line printer as the output device. (Image © reprinted from Rosenfeld and Pfaltz, Pattern Recognition (Oxford: Pergamon Press, 1968), 1:33-61. Used with permission from Elsevier)*

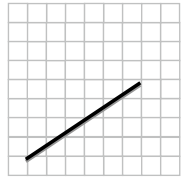
Euclidean or Cartesian Distance Metrics

The Euclidean distance metrics include basic Euclidean geometry identities in Cartesian coordinate spaces; in general, these are simple and obvious to use.

Euclidean Distance

This is the simple distance between two points.

$$EuclideanDistance\left[\{a,b\},\{x,y\}\right] = \sqrt{(a-x)^2+(b-y)^2}$$



Squared Euclidean Distance

This is faster to compute, and omits the square root.

$$SquaredEuclideanDistance\left[\{a,b\},\{x,y\}\right] = (a-x)^2 + (b-y)^2$$

Cosine Distance or Similarity

This is angular distance, or the normalized dot product between two vectors to yield similarity of vector angle; also useful for 3D surface normal and viewpoint matching.

$$cos(\theta) = \frac{A.B}{\|A\| \, \|B\|}$$

$$CosineDistance\left[\{a,b\},\{x,y\}\right] = 1 - \frac{ax+by}{\sqrt{a^2+b^2}\sqrt{x^2+y^2}}$$

Sum of Absolute Differences (SAD) or L1 Norm

The difference between vector elements is summed and taken as the total distance between the vectors. Note that SAD is equivalent to Manhattan distance.

$$\mathrm{SAD}(d_1,d_2) = \sum_{i=0}^{n1}\sum_{j=0}^{n2}\left(d_1[i,j] - d_2[i,j]\right)$$

Sum of Squared Differences (SSD) or L2 Norm

The difference between vector elements is summed and squared and taken as the total distance between the vectors; commonly used in video decoding for motion estimation.

$$\mathrm{SSD}(d_1,d_2) = \sum_{i=0}^{n1}\sum_{j=0}^{n2}\left(d_1[i,j] - d_2[i,j]\right)^2$$

Correlation Distance

This is the correlation difference coefficient between two vectors, similar to cosine distance.

$$C[u,v]=\frac{1-(u-Mean[u]).(v-Mean[v])}{\|u-Mean[u]\|\|v-Mean[v]\|}$$

$$\left[\{a,b\},\{x,y\}\right]=\frac{\left(a+\frac{1}{2}(-a-b)\right)\left(x+\frac{1}{2}(-x-y)\right)+\left(\frac{1}{2}(-a-b)+b\right)\left(\frac{1}{2}(-x-y)+y\right)}{\sqrt{Abs\left[a+\frac{1}{2}(-a-b)\right]^2+Abs\left[\frac{1}{2}(-a-b)+b\right]^2}\sqrt{Abs\left[x+\frac{1}{2}(-x-y)\right]^2 Abs\left[\frac{1}{2}(-x-y)+y\right]^2}}$$

Hellinger Distance

An effective alternative to Euclidean distance, yielding better performance and accuracy for histogram-type distance metrics, as reported in the ROOTSIFT [178] optimization of SIFT. Hellinger distance is defined for L1 normalized histogram vectors as:

$$H(x,y)=\sum_{i=1}^{n}\sqrt{x_i y_i}$$

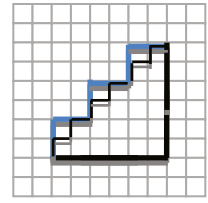
Grid Distance Metrics

These metrics calculate distance analogous to paths on grids. Therefore the distance is measured as grid steps.

Manhattan Distance

Also known as city block difference or rectilinear distance, this measures distance via the route along a grid; there may be more than one path along a grid with equal distance.

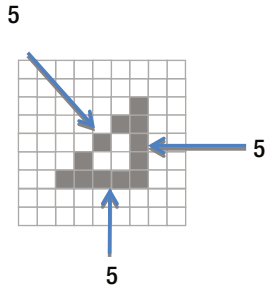
$$ManhattanDistance\left[\{a,b\},\{x,y\}\right]=Abs(a-x)+Abs(b-y)$$



Chebyshev Distance

Also known as chessboard difference, this measures the greatest difference along a grid between two vectors. Note in the illustration below that each side of the triangle would have a Chebyshev distance, or length of 5, but in Euclidean space, one of the lines, the hypotenuse, is longer than the others.

$$ChebyshevDistance\left[\{a,b\},\{x,y\}\right] = Max[Abs(a-x), Abs(b-y)]$$



Statistical Difference Metrics

These metrics are based on statistical features of the vectors, and therefore the distance metrics need not map into a Euclidean space.

Earth Movers Distance (EMD) or Wasserstein Metric

Earth movers distance measures the cost to transform a multidimensional vector, such as a histogram, into another vector. The analogy is an earth mover (bulldozer) moving dirt between two groups of piles to make the piles of dirt in each group the same size. The EMD assumes there is a ground distance between the features in the vector—for example, the distance between bins in a histogram. The EMD is computed to be the minimal cost for the transform, which integrates the distance moved d * the amount moved f, subject to a few constraints [130].

$$COST(P,Q,F) = \sum_{i=1}^{m}\sum_{j=1}^{n} d_{ij} f_{ij}$$

Once the cost is computed, the result is normalized.

$$EMD(P,Q) = \sum_{i=1}^{m}\sum_{j=1}^{n} d_{ij} f_{ij} \sum_{i=1}^{m}\sum_{j=1}^{n} f_{ij}$$

The EMD has a high compute cost and can be useful for image analysis, but EMD is not an efficient metric for feature matching.

Mahalanobis Distance

Also known as quadratic distance, this computes distance using mean and covariance; it is scale invariant.

$$d_{ij} = \left(\left(\bar{x}_i - \bar{x}_j \right)^T S^{-1} \left(\bar{x}_i - \bar{x}_j \right) \right)^{\frac{1}{2}}$$

$$\mathrm{SSD}(\mathrm{d}_1, \mathrm{d}_2) = \sum_{i=-n1}^{n1} \sum_{j=-n2}^{n2} f\left((\mathrm{x}+\mathrm{i}, \mathrm{y}+\mathrm{j}) - \mathrm{g}(\mathrm{x}+\mathrm{i}-\mathrm{d}_1, \mathrm{y}_\mathrm{j}-\mathrm{d}_2) \right)^2$$

where $\bar{x}_i$ = mean of feature vector 1, and $\bar{x}_j$ = mean of feature vector 2.

Bray Curtis Distance

This is equivalent to a ratio of the sums of absolute differences and sums, such as a ratio of norms of Manhattan distances. Bray Curtis dissimilarity is sometimes used for clustering data.

$$BrayCurtisDistance\left[\{a,b,c\}, \{x,y,z\} \right] = \frac{Abs(a-x) + Abs(b-y) + Abs(c-z)}{Abs(a+x) + Abs(b+y) + Abs(c+z)}$$

Canberra Distance

This measures the distance between two vectors of equal length:

$$CanberraDistance\left[\{a,b\}, \{x,y\} \right] = \frac{Abs(a-x)}{Abs(a) + Abs(x)} + \frac{Abs(b-y)}{Abs(b) + Abs(y)}$$

Binary or Boolean Distance Metrics

These metrics rely on set comparisons and Boolean algebra concepts, which makes this family of metrics attractive for optimization on digital computers.

L0 Norm

The L0 norm is a count of non-zero elements in a vector and is used in the Hamming Distance metric and other binary or Boolean metrics.

$$\|x\|_0 = \#(i \mid x_i \neq 0)$$

Hamming Distance

This measures the binary difference or agreement between vectors of equal length—for example, string or binary vectors. Hamming distance for binary bit vectors can be efficiently implemented in digital computers with either complete machine language instructions or as as an XOR operation followed by a bit count operation. Hamming distance is a favorite for matching local binary descriptors, such as LBP, FREAK, CENSUS, BRISK, BRIEF, and ORB.

String distance: 5 = 0001100111 = compare "Hel**lo**Th**ere**" and "Hel**ps**Th**ing**"

Binary distance: 3 = 10100010 = (**0**1**0**011**1**0) XOR (**1**1**0**011**0**0)

Bit count of (u XOR v)

Jaccard Similarity and Dissimilarity

The ratio of pairwise similarity of a binary set (0,1 or true, false) over the number of set elements. Set 1 below contains two bits with the same pairwise value as Set 2, so the similarity is 2/5 and the dissimilarity is 3/5. Jaccard similarity can be combined with Hamming distance.

Set 1:	{1,0,1,1,0}
Set 2:	{1,1,0,1,1}
Jaccard Similarity:	2 / 5 = .4
Jaccard Dissimilarity:	**3 / 5 = .6**

Descriptor Representation

This section discusses how information is represented in the descriptors, including coordinates spaces useful for feature description and matching, with some discussion of multimodal data and feature pyramids.

Coordinate Spaces, Complex Spaces

There are many coordinate systems used in computer vision, so being able to transform data between coordinate systems is valuable. Coordinate spaces are analogous to basis spaces. Often, choosing the right coordinate system provides advantages for feature representation, computation, or matching. Complex spaces may include multivariate collections of scalar and vector variables, such as gradients, color, binary patterns, and statistical moments of pixel regions. See Figure 4-4.

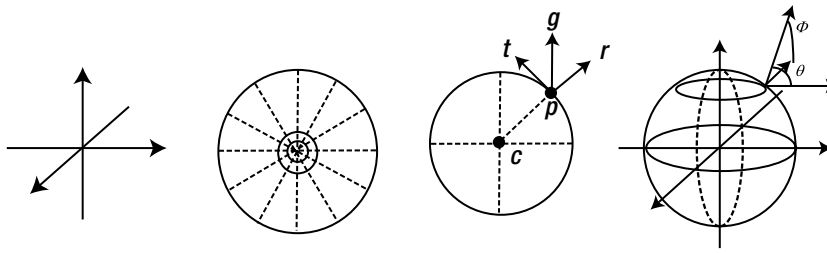


Figure 4-4. *Coordinate spaces, Cartesian, polar, radial, and spherical*

Cartesian Coordinates

Images are typically captured in the time domain in a Cartesian space, and for many applications translating to other coordinate spaces is needed. The human visual system views the world as a complex 3D spherical coordinate space, and humans can, through a small miracle, map the 3D space into approximate or relative Cartesian coordinates. Computer imaging systems capture data and convert it to Cartesian coordinates, but depth perception and geometric accuracy are lost in the conversion. (Chapter 1 provided a discussion of depth-sensing methods and 3D imaging systems, including geometric considerations.)

Polar and Log Polar Coordinates

Many descriptors mentioned later in Chapter 6 use a circular descriptor region to match the human visual system. Therefore, polar coordinates are logical candidates to bin the feature vectors. For example, the GLOH [144] method uses polar coordinates for histogram gradient binning, rather than Cartesian coordinates as used in the original SIFT [161] method. GLOH can be used as a retrofit for SIFT and has proved to increase accuracy [144]. Since the circular sampling patterns tend to provide better rotational invariance, polar coordinates and circular sampling are a good match for descriptor design.

Radial Coordinates

The RIFF descriptor (described later in Chapter 6) uses a local radial coordinate system to describe rotationally invariant gradient-based feature descriptors. The radial coordinate system is based on a radial gradient transform (RGT) that normalizes vectors for invariant binning.

As shown in Figures 4-4 and 6-27, the RGT creates a local coordinate system within a patch region *c,* and establishes two orthogonal basis vectors (*r,t*) relative to any point *p* in the patch, *r* for the radial vector, and *t* for the tangential vector. The measured gradients *g* at all points *p* are projected onto the radial coordinate system (*r,t*), so that the gradients are represented in a locally invariant fashion relative to the interest point *c* at the center of the patch. When the patch is rotated about c, the gradients rotate also, and the invariant representation holds.

Spherical Coordinates

Spherical coordinates, also referred to as 3D polar coordinates, can be applied to the field of 3D imaging and depth sensing to increase the accuracy for description and analysis. For example, depth cameras today typically only provide (*x,y)*, and *Z* depth information for each sample. However, this is woefully inadequate to describe the complex geometry of space, including warping, radial distortion and nonlinear distance between samples. Chapter 1 discussed the complexities of 3D space, depth measurements, and coordinate systems.

Gauge Coordinates

The G-SURF methods [188] use a differential geometry concept [190] of a local region Gauge coordinate system to compute the features. Gauge coordinates are local to the image feature, and they carry advantages for geometrical accuracy. Gauge derivatives are rotation and translation invariant.

Multivariate Spaces, Multimodal Data

Multivariate spaces combine several quantities, such as Tensor spaces which combine scalar and vector values, and are commonly used in computer vision. While raw image data may be scalar values only, many feature descriptors compute local gradients at each pixel, so the combination of pixel scalar value and gradient vector forms a tensor or multivariate space. For example, color spaces (see Chapter 2) may represent color as a set of scalar and vector quantities, such as the hue, saturation, and value (HSV) color space illustrated in Figure 2-9, where the vectors include ***HS*** with ***H*** hue as the vector angle and ***S*** saturation as the vector magnitude.***V*** is another vector with two purposes, first as the axis origin for the ***HS*** vector and second as the color intensity or gray scale vector ***V***. It is often useful to convert raw *RGB* data into such color spaces for ease of analysis—for example, to be able to uniformly change the color intensity of all colors together so as to affect brightness or contrast.

In general, by increasing the dimensions of the feature space, more discrimination and robustness can be added. For example, the LBP pattern as described later in Chapter 6 can be extended into multiple variables by adding features such as a rotational invariant representation (RILBP); or by replicating the LBP across *RGB* color cannels as demonstrated in the color LBP descriptor; or by extending the LBP pattern into spatio-temporal 3-space, like the LBP-TOP to add geometric distortion invariance.

Multimodal sensor data is becoming widespread with the proliferation of mobile devices that have built-in GPS, compass, temperature, altimeter, inertial and other sensors. An example of a multimodal, multivariate descriptor is the SIFT-GAFD [245] method, as illustrated in Figure 4-5, which adds accelerometer information in the form of a gravity vector to the SIFT descriptor. The gravity vector is referred to as *global orientation*, and the SIFT local pixel region gradient is referred to as the *local orientation*.

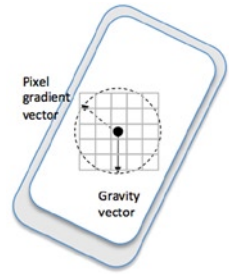


Figure 4-5. *Multimodal descriptor using accelerometer data in the form of a gravity vector, in a feature descriptor as used in the SIFT-GAFD method [245]. The gravity vector of global orientation can be used for feature orientation with respect to the environment*

Feature Pyramids

Many feature descriptors are computed in a mono-scale fashion using pixel values at a given scale only, and then for feature detection and matching the feature is searched for in a scale space image pyramid. However, by computing the descriptor at multiple scales and storing multiple scaled descriptors together in a *feature pyramid*, the feature can be detected on mono-scale images with scale variance without using a scale space pyramid.

For interest point and feature descriptor methods, *scale invariance* can be addressed either by: (1) scaling the images prior to searching, as in the scale space pyramid methods discussed later in this chapter; or (2) scaling and pyramiding multiple scales of the feature in the descriptor. Shape-based methods are by nature more scale invariant than interest point and feature descriptor methods, since shape-based methods depend on larger polygon structures and shape metrics.

Descriptor Density

Depending on the image data, there will be a different number of good interest points and features, since some images have more pronounced texture. And depending on the detector method used, images with high texture structure, or wider pixel intensity range differences, will likely generate more interest points than images with low contrast and smooth texture.

A good rule of thumb is that between .1 and 1 percent of the pixels in an image can yield raw, unfiltered interest points. The more sensitive detectors such as FAST and the Harris detector family are at the upper end of this range (see Appendix A). Of course, detector parameters are tuned to reduce unwanted detection for each application.

Interest Point and Descriptor Culling

In fact, even though the interest point looks good, the corresponding descriptor computed at the interest point may not be worth using and will be discarded in some cases. Both interest points and descriptors are culled. So tuning the detector and

descriptor together are critical trial-and-error processes. Using our base assumption of .1 to 1 percent of the pixels yielding valid raw interest points, we can estimate the possible detected interest points based on video resolution, as shown in Table 4-2.

Table 4-2. *Possible Range of Detected Interest Points per Image*

	480p NTSC	1080p HD	2160p 4kUHD	4320p 8kUHD
Resolution	640 x 480	1920 x 1080	3840 × 2160	7680 x 4320
Pixels	307200	2073600	8294400	33177600
Interest points	300 – 3k	2k – 21k	8k – 83k	33k – 331k

Depending on the approach, the detector may be run only at mono-scale or across a set of scaled images in an image pyramid scale space. For scale space search methods, the interest point detector is run at each pixel in each image in the pyramid. What methods can be used to cull interest points to reduce the interest point density to a manageable number?

One method to select the best interest points is to use an *adaptive detector tuning method* (as discussed in Chapter 6 under "Interest Point Tuning"). Other approaches include only choosing interest points at a given threshold distance apart—for example, an interest point that cannot be adjacent to another interest point within a five-pixel window, with the best candidate point selected within the threshold.

Another method is to vary the search strategy as discussed in this chapter—for example, search for features at a lower resolution of the image pyramid, identify the best features, and record their positions, and perhaps search at higher levels of the pyramid to confirm the feature location, then compute the descriptors. This last method has the drawback of missing fine-grain features by default, since features may only be present at full image resolution.

Yet another method is to look for interest points every other pixel or within grid-sized regions. All of the above methods are used in practice, and other methods exist besides.

Dense vs. Sparse Feature Description

A *dense* descriptor makes use of all the pixels in the region or patch. By "dense" we mean that the kernel sampling pattern includes all the pixels, since a sparse kernel may select specific pixels to use or ignore. SIFT and SURF are classic examples of dense descriptors, since all pixels in rectangular regions contribute to the descriptor computation.

Many feature description methods, especially local binary descriptor methods, are making use of *sparse patterns*, where selected pixels are used from a region rather than all the pixels. The FREAK descriptor demonstrates one of the most ingenious methods of sparse sampling by modeling the human visual system, using a circular search region, and leveraging the finer resolution sampling closer to the center of the region, as well as tuning a hierarchy of local sampling patterns of increasing resolution for optimal results. Not only can sparse features potentially use less memory and reduce computations, but the sparse descriptor can be spread over a wider area to compensate for feature anomalies that occur in smaller regions.

Descriptor Shape Topologies

For this discussion, we view descriptor shape *topology* with an eye toward surveying the various shapes of the pixel regions used for descriptor computations. Part of the topology is the shape or boundary, and part of the topology is the choice of dense vs. sparse sampling patterns, discussed later in this chapter. Sampling and pattering methods range from the simple rectangular regions up to the more complex sparse local binary descriptor patterns. As will be discussed in Chapter 6, both 2D and 3D descriptors are being designed to use a wide range of topologies. Let's look at a few topological design considerations, such as patch shape, sub-patches, strips, and deformable patches.

Which shape is better? The answer is subjective and we do not attempt to provide absolute answers, just offer a survey.

Correlation Templates

An obvious shape is the simple rectangular regions commonly used by correlation template matching methods. The descriptor is thus the *mugshot,* or actual image in the template region. To select sub-spaces within the rectangle, a mask can be used—for example, it could be a circular mask inside the bounding rectangle to mask off peripheral pixels from consideration.

Patches and Shape

The literature commonly refers to the feature shape as a *patch*, and usually a rectangular shape is assumed. Patch shapes are commonly rectangular owing to the ease of coding 2D array memory access. Circular patches are widely used in the local binary descriptor methods.

However, many descriptors also compute features *over multiple patches* or regions, not just a single patch. Here are some common variations on patch topology.

Single Patches, Sub-Patches

Many descriptors limit the patch count to a single 2D patch. This is true of most common descriptors that are surveyed in Chapter 6. However, some of the local binary descriptors use a set of integral image *sub-patches* at specific points within the larger patch—for example, BRIEF uses a 5x5 integral image sub-patch at each sample point in the local binary pattern, within the larger 31x31 pixel patch region, so the value of each sub-patch becomes the value used for the point-pair comparison. The goal is to filter the values at each point to remove noise.

Deformable Patches

Rather than use a rigid shape, such as a fixed-size rectangle or a circle, feature descriptors can be designed with deformation in mind, such as scale deformations [345,346], and affine or homographic deformation [220], to enable more robust matching. Examples

include the DeepFlow [344,394] deep matching method, and RFM2.3, as will be discussed in Chapter 6. Also, the D-NETS [135] method, using the fully connected or sparse connected topology, can be considered to be deformable in terms of invariance of the placement of the strip patterns; see Figure 4-7 and the discussion of D-nets in Chapter 6. Many feature learning methods discussed later in this chapter also use deformed features for training.

Fixed descriptor shapes, such as rigid rectangles and circles, can detect motion under a rigid motion hypothesis, where the entire descriptor is expected to move with some amount of variance, such as in scale or affine transformation. However, for activity recognition and motion, a more deformable descriptor model is needed, and DeepFlow [344,394] bridges the gap between descriptor matching methods and optical flow matching methods, using deformable patches and deep matching along the lines of deep learning networks.

Multi-Patch Sets

The SIFT descriptor uses multi-patch sets of three patches from adjacent DoG images taken from the scale space pyramid structure, as shown in Figure 6-15. Several other methods, such as the LBP-TOP and VLBP shown in Figure 6-12, use sets of patches spread across a volume structure. LBP-TOP uses patches from adjacent planes, and the VLBP uses intersecting patches in 3-space. Dynamic texture methods use sets of three adjacent patches from spatio-temporal image frame sets, as frame n-2, frame n-1, and frame-0 (current frame).

TPLBP, FPLBP

The three-patch LBP TPLBP and four-patch LBP FPLBP [244] utilize novel multi-patch sampling patterns to add sparse local structure into a composite LBP descriptor. As shown in Figure 4-6, the three-patch LBP uses a radial set of LBP patterns composed using alternating sets of three patches, and the four-patch LBP uses a more distributed pairing of patches over a wider range.

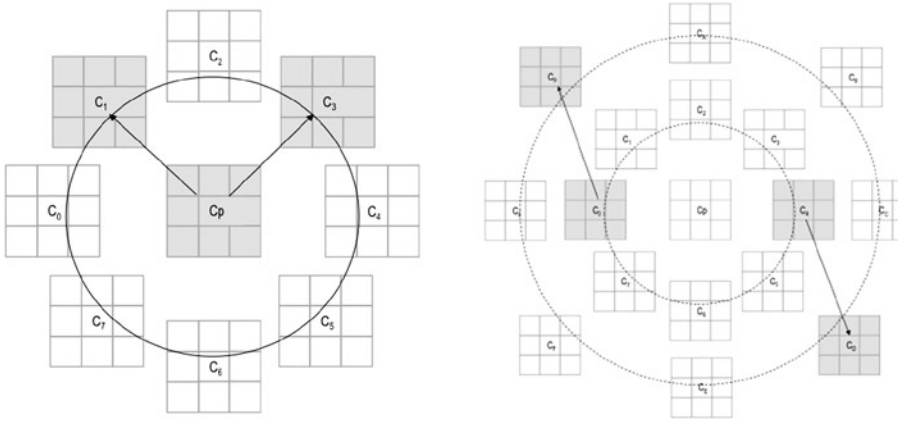


Figure 4-6. *Novel multi-patch sets developed by Wolf et. al [244]. (Left) The TPLBP compares the values from three-patch sets around the ring to compute the LBP code, eight sets total, so there is one set for each LBP bit. (Right) The four-patch LBP uses four patches to computed bits using two symmetrically distributed patches from each ring, to produce each bit in the LBP code. The radius of each ring is a variable, the patch pairing is a variable, and the number of patches per ring is a variable; here, there are eight patches per ring*

Strip and Radial Fan Shapes

Radial fans or spokes originating at the feature interest point location or shape centroid can be used as the descriptor sampling topology—for example, with Fourier shape descriptors (as discussed in Chapter 6; see especially Figure 6-29).

D-NETS Strip Patterns

The D-NETS method developed by Hundelshausen and Sukthankar[135] uses a connected graph-shaped descriptor pattern with variations in the sampling pattern possible. The authors suggest that the method is effective using three different patterns, as shown in Figure 4-7:

1. Fully connected graph at interest points
2. Sparse or iterative connected graph at interest points
3. Densely sampled graph over a chosen grid

Figure 4-7. *Reduced resolution examples of the D-NETS [135] sampling patterns. (Left) Full dense connectivity at interest points. (Center) Sparse connectivity at interest points. (Right) Dense connectivity over a regular sampling grid. The D-NETS authors note that a dense sampling grid with 10 pixel spacing is preferred over sampling at interest points*

The descriptor itself is composed of a set of *d-tokens,* which are strips of raw pixel values rather than a value from a patch region: the strip is the region, and various orientations of lines are the pattern. The sampling along the strip is between 80 and 20 percent of the strip length rather than the entire length, omitting the endpoints, which is claimed to reduce the contribution of noisy interest points. The sampled points are combined into a set ***s*** of uniform chunks of pixels and normalized and stored into a discrete d-token descriptor.

Object Polygon Shapes

The object and polygon shape methods scan globally and regionally to find the shapes in the entire image frame or region. The goal is to find an object or region that is cohesive. A discussion of the fundamental methods for segmentation polygon shapes for feature descriptors is provided here, including:

- Morphological object boundary methods
- Texture or regional structural methods
- Superpixel or pixel similarity methods
- Depth map segmentation

Chapter 6 provides details on a range of object shape factors and metrics used to statistically describe the features of polygon shape. Note that this topic is often discussed in the literature as “image moments”; a good source of information is Flusser et.al.[518].

Morphological Boundary Shapes

One method for defining polygon shapes is to use morphology. Morphological segmentation is a common method for region delineation, either as a binary object or as a gray scale object. Morphological shapes are sometimes referred to as *blobs*. In both binary and gray scale cases, thresholding is often used as a first step toward defining the

object boundary, and morphological reshaping operations such as ERODE and DILATE are used to grow, shrink, and clean up the shape boundary. Morphological segmentation is threshold- and edge-feature driven. (Chapter 3 provided a discussion of the methods used for morphology and thresholding.)

Texture Structure Shapes

Region texture is also used to segment polygon shapes. Texture segmentation is a familiar image-processing method for image analysis and classification, and is an ideal method for segmentation in a nonbinary fashion. Texture reveals structure that simple thresholding ignores. As shown in Figure 6-6, the LBP operator can detect local texture, and the texture can be used to segment regions such as sky, water, and land. Texture segmentation is based on local image pixel relationships. (Several texture segmentation methods were surveyed in Chapter 3.)

Super-Pixel Similarity Shapes

Segmenting a region using super-pixel methods is based on the idea of collapsing similar pixels together—for example, collapsing pixels together with similar colors into a larger shape. The goal is to segment the entire image region into super-pixels. Super-pixel methods are based on similarity. (Several super-pixel processing methods were discussed in Chapter 3.)

Local Binary Descriptor Point-Pair Patterns

Local binary descriptor shapes and sampling patterns, such as those employed in FREAK, BRISK, ORB, and BRIEF, are good examples to study in order to understand the various tradeoffs and design approaches. We will examine local binary shape and pattern concepts here. (Chapter 6 provides a more detailed survey of each descriptor.)

Local binary descriptors use a *point-pair sampling method*, where pairs of pixels are assigned to each other for a binary comparison. Note that a drawback of local binary descriptors and point-pair comparisons is that small changes in the image pixel values in the local region may manifest as *binary artifacts*. Seemingly insignificant changes in a set of pixel values may cause problems during matching that are pronounced for: (1) noisy images, and (2) images with constant gray level. However, each local binary descriptor method attempts to mitigate the binary artifact problems. For example, BRISK (see Figure 4-10 later) and ORB (see Figure 4-11 later) compute a *filtered region* surrounding each interest point to reduce the noise component prior to the binary comparison.

Another method to mitigate the binary artifact problem of constant gray level is used in a modification of the LBP method called the local trinary pattern operator, or LTP [522] (see also reference[173], Section 2.9.3), which uses *trinary values* of *{-1, 0,1}* to describe regions. A threshold band is established for the LTP to describe near-constant gray values as 0, values above the threshold band as 1, and values below the threshold band as -1. The LTP can be used to describe both smooth regions of constant gray level and contrasted regions in the standard LBP. In addition, the compare threshold for point-pairs can be tuned to compensate for noise, illumination, and contrast, as employed in nearly all local binary descriptor methods.

Figure 4-8 (left image) illustrates a hypothetical descriptor pattern to include selected pixels as the black values, while the center left image shows a strip-oriented shape and pattern where the descriptor calculates the descriptor over pixels along a set of line segments with no particular symmetry like the DNETS [135] method.

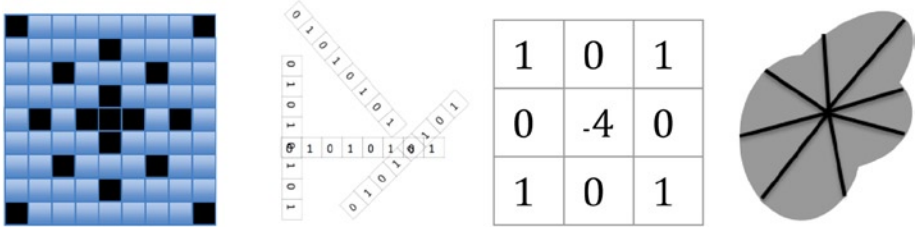


Figure 4-8. *illustrating various descriptor patterns and shapes. (Left) Sparse. (Center left) Nets or strips. (Center right) Kernels. (Right) Radial spokes*

In Figure 4-8 also, the center right image illustrates a convolution kernel where the filter shape and filter goal are specified, while the right image is a blob shape using radial pixel sampling lines originating at the shape centroid and terminating on the blob perimeter. Note that a 1D Fourier descriptor can be computed from an array containing the length of each radial line segment from the centroid to the perimeter to describe shape, or just an array of raw pixel values can be kept, or else D-nets can be computed.

A feature descriptor may be designed by using one or more shapes and patterns together. For example, the hypothetical descriptor pattern in Figure 4-8 (left image) uses one pattern for pixels close to the interest point, another pattern uses pixels farther away from the center to capture circular pattern information, and another pattern covers a few extrema points. An excellent example of tuned sampling patterns is the FREAK descriptor, discussed next.

FREAK Retinal Patterns

The FREAK [130] descriptor shape, also discussed in some detail in Chapter 6, uses local binary patterns based on the human retinal system, as shown in Figure 4-9, where the density of the receptor cells in the human visual system is greater in the center and decreases with distance from center. FREAK follows a similar pattern when building the local binary descriptors, referred to as a *coarse-to-fine* descriptor pattern, with fine detail in the center of the patch and coarse detail moving outward. The coarse-to-fine method also allows for the descriptor to be matched in coarse-to-fine segments. The coarse part is matched first, and if the match is good enough, the fine feature components are matched as well.

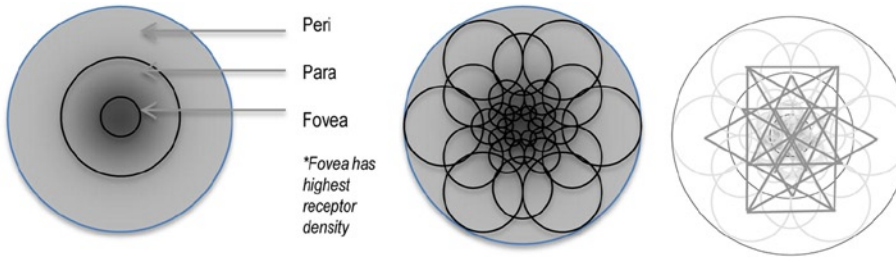


Figure 4-9. *(Left) The human visual system concentration of receptors in the center Fovea region with less receptor density moving outward to periphery vision regions of Para and Peri. (Center) FREAK [130] local binary pattern sampling regions, six regions in each of six overlapping distance rings from the center, size of ring denotes compare point averaging area. (Right) Hypothetical example of a FREAK-style point-pair pattern*

FREAK descriptors can be built with several patterns within this framework. For FREAK, the actual pattern shape and point-pairing are designed during a training phase where the best point-pair patterns are learned using a method similar to ORB [134] to find point-pairs with high variance. The pattern is only constrained by the training data; only 45 point-pairs are used from the 32x31 image patch region.

As illustrated in Figure 4-9, the pairs of points at the end of each line segment are compared, the set of compare values are composed into a binary descriptor vector using 16 bytes, and a cascade of four separate 16-byte coarse-to-fine patterns are included in the descriptor set. Typically, the coarse pattern alone effectively rejects bad matches, and the finer patterns are used to qualify only the closest matches.

Brisk Patterns

The BRISK descriptor [131] point-pair sampling shape is symmetric and circular, composed of 60 total points arranged in four concentric rings, as shown in Figure 4-10. Surrounding each of the 60 points is a sampling region shown in blue, the sampling regions increase in size with distance from the center, and also proportional to the distance between sample points. Within the sampling regions, Gaussian smoothing is applied to the pixels and a local gradient is calculated over the smoothed region.

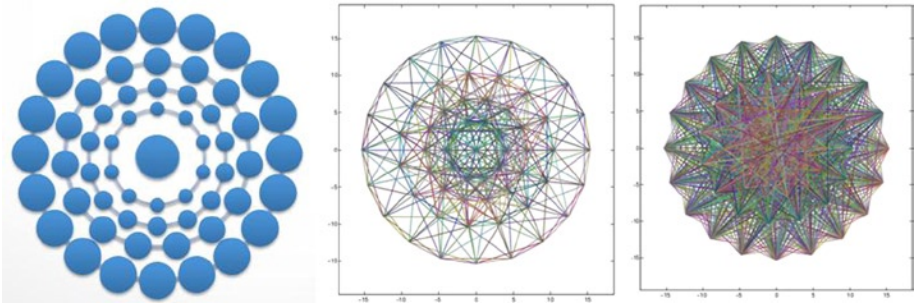


Figure 4-10. *(Left) BRISK concentric sampling grid pattern. (Center) Short segment pairs. (Right) Long distance pairs. Note that the size of the region (left image) for each selected point increases in diamter with distance from the center, and the binary comparison is computed from the center point of each Gaussian-sampled circular region, rather than from each solitary center point. (Center and right images used by permission © Josh Gleason[143])*

Like other local binary descriptors, BRISK compares pairs of points to form the descriptor. The point- pairs are specified in two groups: (1) *long segments,* which are used together with the region gradients to determine angle and direction of the descriptor, the angle is used to rotate the descriptor area, and then the pair–wise sampling pattern is applied;(2) *short segments,* which can be pair-wise compared and composed into the 512-bit binary descriptor vector.

ORB and BRIEF Patterns

ORB [134] is based in part on the BRIEF descriptor [132,133], thus the name **Or**iented **B**rief, since ORB adds orientation to the BRIEF method and provides other improvements as well. For example, ORB also improves the interest point method by qualifying FAST corners using Harris corner methods, and improves corner orientation using Rosin's method [61] in order to steer the BRIEF descriptor to improve rotational invariance (BRIEF is known to be sensitive to rotation).

ORB also provides a very good point-pair training method, which is an improvement over BRIEF. In BRIEF, as shown in Figure 4-11, the sample points are specified in a random distribution pattern based on a Gaussian distribution about the center point within the 31x31 patch region; the chosen number of sample points is 256. Selected sample point-pairs are compared to each other to form the binary descriptor vector. The value of each point is calculated via an integral image method to smooth a 5x5 region into the point value.

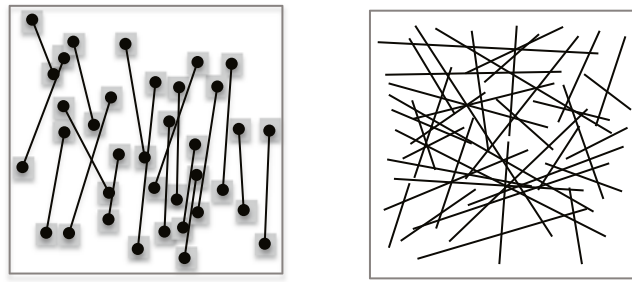


Figure 4-11. *(Left) An ORB style pattern at greatly reduced point pair count resolution, using < 32 points instead of the full 256 points. (Right) A BRIEF style pattern using randimized point-pairs*

To learn the descriptor point-pair sample and compare pattern, ORB uses a training algorithm to find uncorrelated points in the training set with high variance, and selects the best 256 points to define the pairwise sampling patterns used to create the binary feature vector. So the shape and pattern are nonsymmetric, as shown in Figure 4-11, similar to some DNETS patterns. The ORB point-pair patterns are dependent on the training data.

Note in Figure 4-11 that a BRIEF style pattern (right image) uses random point-pairs. Several methods for randomizing point-pairs are suggested by the developers [132]. The ORB pattern shown in Figure 4-11 is based on choosing point-pairs that have high statistical variance within a bounding 31x31 image patch, where the smaller 5x5 gray image patch regions are centered at the chosen interest points. Then each 5x5 region is smoothed using an integral image method to yield a single value for the point.

Descriptor Discrimination

How discriminating is a descriptor? By *discrimination* we mean how well the descriptor can uniquely describe and differentiate between other features. Depending on the application, more or less discrimination is needed, thus it is possible to *over-describe* a feature by providing more information and invariance than is useful, or to *under-describe* the feature by limiting the robustness and invariance attributes. Feature descriptor discrimination for a given set of robustness criteria may be important and interesting, but discrimination is not always the right problem to solve in some cases.

The need for increased discrimination in the descriptor can be balanced in favor of using a cascade of simple descriptors like correlation templates under the following assumptions.

1. **Assuming cheap massively parallel compute**, deformable descriptors such as Taylor and Rosin's RFM2.3 [220] become a more attractive option, allowing simple weakly discriminating correlation templates or pixel patches to be used and deformed in real-time in silicon using the GPU texture sampler for scale, affine and homographic transforms. Matching and correspondence under various pose variations and lighting variations can be easily achieved using parallel GPU SIMT/SIMD compute and convolution kernels. So, the GPU can effectively allow a simple correlation patch to be warped and contrast enhanced to be used as a deformable descriptor and compared against target features.
2. **Assuming lots of fast and cheap memory**, such as large memory cache systems, many nondiscriminating descriptors or training patterns can be stored in the database in the memory cache. Various weighting schemes such as those used in neural networks and convolutional networks can be effectively employed to achieve desired correspondence and quality. Also, other boosting schemes can be employed in the classifier, such as the Adaboost method, to developed strong classifiers from weakly discriminating data.

In summary, both highly discriminating feature descriptors and cascades of simple weakly discriminating feature descriptors may be the right choice for a given application, depending on the target system.

Spectra Discrimination

One dimension of feature discrimination is the chosen descriptor spectra or values used to represent the feature. We refer to *spectra* simply as values within a spectrum or over a continuum. A feature descriptor that only uses a single spectra, such as a histogram of intensity values, will have discrimination to intensity distributions, with no discrimination for other attributes such as shape or affine transforms. For example, a feature descriptor may increase the level of discrimination by combining a multivariate set of spectra such as *RGB* color, depth, and local area gradients of color intensity.

It is well known [248] that the human visual system discriminates and responds to gradient information in a scale and rotationally invariant manner across the retina, as demonstrated in SIFT and many other feature description methods. Thus the use of gradients is a common and preferred spectra for computer vision.

Spectra may be taken over a range of variables, where simple scalar ranges of values are only one type of spectra:

1. Gray scale intensity
2. Color channel intensity
3. Basis function domains (frequency domain, HAAR, etc.)
4. 2D or 3D gradients

5. 3D surface normals
6. Shape factors and morphological measures
7. Texture metrics
8. Area integrals
9. Statistical moments of regions
10. Hamming codes from local binary patterns

Each of the above spectra types, along with many others that could be enumerated, can be included in a multivariate feature descriptor to increase discrimination. Of course, discrimination requirements for a chosen application will guide the design of the descriptor. For example, an application that identifies fruit will be more effective using color channel spectra for fruit color, shape factors to identify fruit shapes, and texture metrics for skin texture.

One way to answer the question of discrimination is to look at the information contained in the descriptor. Does the descriptor contain multivariate collections of spectra, and how many invariance attributes are covered, such as orientation or scale?

Region, Shapes, and Pattern Discrimination

Shape and pattern of the feature descriptor are important dimensions affecting discrimination. Each feature shape has advantages and disadvantages depending on the application. Surprisingly, even a single pixel can be used as a feature descriptor shape (see Figure 1-7). Let's look at other dimensions of discrimination.

Shapes and patterns may be classified as follows:

1. A single pixel (discussion of single pixel description methods to follow)
2. A line of pixels
3. A rectangular region of pixels
4. A polygon shape or region of pixels
5. A pattern or set of unconnected pixels, such as foveal patterns

The shape of the descriptor determines attributes of discrimination. For example, a rectangular descriptor will be limited in the rotational invariance attribute compared to a circular shaped descriptor. Also, a smaller shape for the descriptor limits the range to a smaller area, and also limits scale invariance. A larger size descriptor area carries more pixels which can increase discrimination.

Descriptor shape, pixel sampling pattern, sampling region size, and pixel metrics have been surveyed by several other researchers [128–130]. In this section, we dig deeper and wider into specific methods used for feature descriptor tuning, paying special attention to local binary feature descriptors, which hold promise for low power and high performance.

Geometric Discrimination Factors

The shape largely determines the amount of rotational invariance possible. For example, a rectangular shape typically begins to fall off in rotational discrimination at around 15 degrees, while a circular pattern typically performs much better under rotational variations. Note that any poorly discriminating shape or pattern descriptor can be enhanced and made more discriminating by incorporating more than one shape or pattern into the descriptor vector.

A shape and pattern such as a HAAR wavelet, as used in the Viola Jones method, integrates all pixels in a rectangular region, yielding the composite value of all the pixels in the region. Thus there is no local fine-detail pattern information contained in the descriptor, leading to very limited local area discrimination and poor rotational invariance or discrimination.

Another example of poor rotational discrimination is the rectangular correlation template method, which compares two rectangular regions pixel by pixel. However, several effective descriptor methods use a rectangular-shaped region.

In general, rectangles are a limitation to rotational invariance. However, SURF uses a method of determining the dominant orientation of the rectangular HAAR wavelet features within a circular neighborhood to achieve better rotational invariance. And SIFT uses a method to improve rotational invariance and accuracy by applying a circular weighting function to the rectangular regions during the binning stage.

It should also be noted that descriptors with low discrimination are being used very effectively in targeted applications, such as correlation methods for motion estimation in video encoding. In this case, the rectangle shape is a great match to the encoding problem and lends itself to highly optimized fixed function hardware implementations, since frame-to-frame motion can be captured very well in rectangular regions, and there is typically little rotation or scale change from frame to frame for at 30 Hz frame rates, just translation.

With this discussion in mind, descriptor discrimination should be *fitted* appropriately to the application, since adding discrimination comes at a cost of compute and memory.

Feature Visualization to Evaluate Discrimination

Another way to understand discrimination is to use the feature descriptor itself to reconstruct images from the descriptor information alone, where we may consider the collection of descriptors to be a compressed or encoded version of the actual image. Image compression, encoding, and feature description are related; see Figure 3-18. Next, we examine a few examples of image reconstruction from the descriptor information alone.

Discrimination via Image Reconstruction from HOG

Figure 4-12 visualizes a reconstruction using the HOG descriptor [106]. The level of detail is coarse and follows line and edge structure that matches the intended use of HOG. One key aspect of the discrimination provided by HOG is that no image smoothing is used on the image prior to calculating the descriptor. The HOG research shows that smoothing the image results in a *loss of discrimination*. Dalal and Triggs[106] highlight their deliberate intention to avoid image smoothing to preserve image detail.

Figure 4-12. *Discrimination via a visualization of the HOG description. (Image (c) Carl Vodrick, used by permission.) See also "HOGgles: Visualizing Object Detection Features, Carl Vondrick, Aditya Khosla, Tomasz Malisiewicz, Antonio Torralba, Massachusetts Institute of Technology, Oral presentation at ICCV 2013"*

However, some researchers argue that noise causes problems when calculating values such as local area gradients and edges, and further recommend that noise be eliminated from the image by smoothing prior to descriptor calculations; this is the conventional wisdom in many circles. Note that there are many methods to filter noise without resorting to extreme Gaussian-style smoothing, convolution blur, and integral images, which distort the image field.

Some of the better noise-filtering methods include speckle removal filters, rank filtering, bilateral filters, and many other methods that were discussed in Chapter 2. If the input image is left as is, or at least the best noise filtering methods are used, the feature descriptor will likely retain more discrimination power for fine-grained features.

Discrimination via Image Reconstruction from Local Binary Patterns

As shown in Figure 4-13, d'Angelo and Alahi[127] provide visualizations of images reconstructed from the FREAK and BRIEF local binary descriptors. The reconstruction is rendered entirely from the descriptor information alone, across the entire image. BRIEF uses a more random pattern to compare points across a region, while FREAK uses a trained and more foveal and symmetrical pattern with increased detail closer to the center of the region. And d'Angelo and Alahi[127] note that the reconstruction results are similar to Laplacian filtered versions of the original image, which helps us understand that the discrimination of these features appears to be structurally related to detailed edge and gradient information.

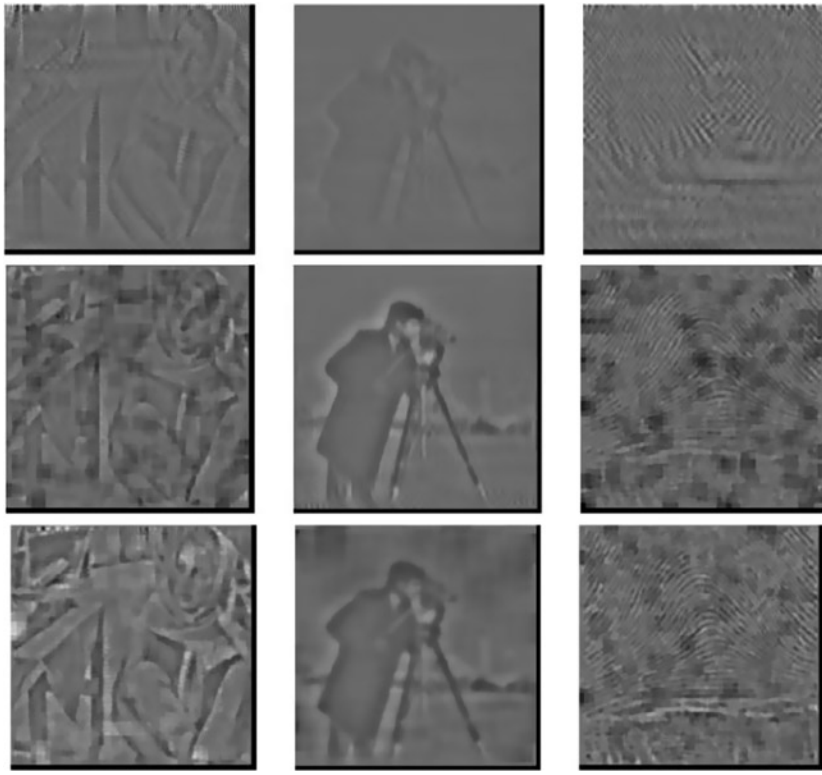


Figure 4-13. *Images reconstructed using local binary descriptors using 512 point-pairs. (Top row) BRIEF. (Middle row) Randomized FREAK (more similar to BRIEF). (Bottom row) Binary FREAK using the foveal pattern Images (c) Alexandre Alahi, used by permission*

The d'Angelo and Alahi reconstruction method [127] creates an image from a set of overlapping descriptor patches calculated across the original image. To reconstruct the image, the descriptors are first reconstructed using a novel method to render patches, and then the patches are merged by averaging the overlapping regions to form an image, where the patch merge size may vary as desired. For example, note that Figure 4-13 uses 32x32 patches for the Barbara images in the left column, and a 64x64 patch size for the cameraman in the center column. Also note that Barbara is not reconstructed with the same discrimination as the cameraman, whose image contains finer details.

Discrimination via Image Reconstruction from SIFT Features

Another method of approximate image reconstruction [105] proves the discrimination capabilities of SIFT descriptors; see Figure 4-14. The reconstruction method for this research starts by taking an unknown image containing a scene such as a famous building, finding the set of Hessian-affine region detectors in the image, extracting associated SIFT feature descriptors, and then saving a set of elliptical image patch regions around the SIFT descriptors.

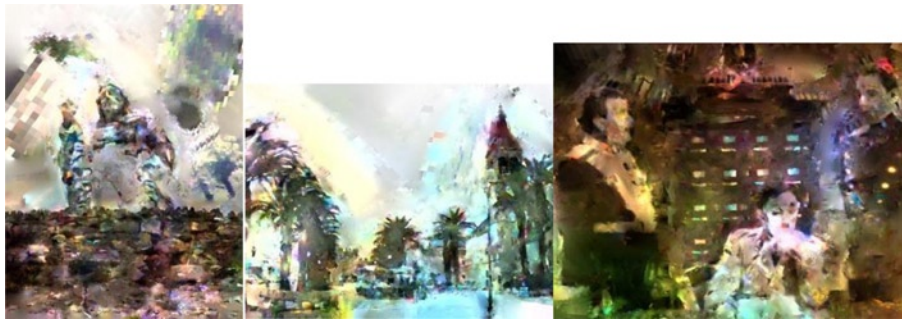


Figure 4-14. *Image reconstruction of common scenes using combined SIFT descriptors taken from several views of the same object, images (c) Herve Jegou, used by permission*

Next, an image database containing similar and, it is hoped, matching images of the same scene are searched to find the closest matching SIFT descriptors at Hessian-affine interest points. Then a set of elliptical patch regions around each SIFT descriptor is taken. The elliptical patches found in the database are warped into a synthesized image based on a priori interest region geometric parameters of the scenes.

The patches are stitched together via stacking and blending overlapping patches and also via smooth interpolation. Any remaining holes are filled by smooth interpolation. One remarkable result of this method is the demonstration that an image can be reconstructed from a set of patches from different images at different orientations, since the feature descriptors are similar; and in this case, the discrimination of the SIFT descriptor is demonstrated well.

Accuracy, Trackability

Accuracy can be measured in terms of specific feature attributes or robustness criteria; see Tables 4-1 and 7-4. A given descriptor may outperform another descriptor in one area and in not another. In the research literature, the accuracy and performance of each new feature descriptor is usually benchmarked against the standby methods SIFT and SURF. The feature descriptor accuracy is measured using commonly accepted ground truth datasets designed to measure robustness and invariance attributes. (See Appendix B for a survey of standard ground truth datasets, and Chapter 7 for a discussion about ground truth dataset design.)

A few useful accuracy studies are highlighted here, illustrating some of the ways descriptor and interest point accuracy can be measured. For instance, one of the most comprehensive surveys of earlier feature detector and descriptor accuracy and invariance is provided by Mikolajczyk and Schmid[144], covering a range of descriptors including GLOH, SIFT, PCA-SIFT, Shape Context, spin images, Hessian Laplacian GLOH, cross correlation, gradient moments, complex filters, differential invariants, and steerable filters.

In Gauglitz et al.[145], there are invariance metrics for zoom, pan, rotation, perspective distortion, motion blur, static lighting, and dynamic lighting for several feature metrics, including Harris, Shi-Tomasi, DoG, Fast Hessian, FAST, and CenSurE, which are discussed in Chapter 6. There are also metrics for a few classifiers, including

randomized trees and FERNS, which are discussed later in this chapter. Figure 4-15 provides some visual comparisons of feature detector and interest point accuracy from Gauglitz [145].

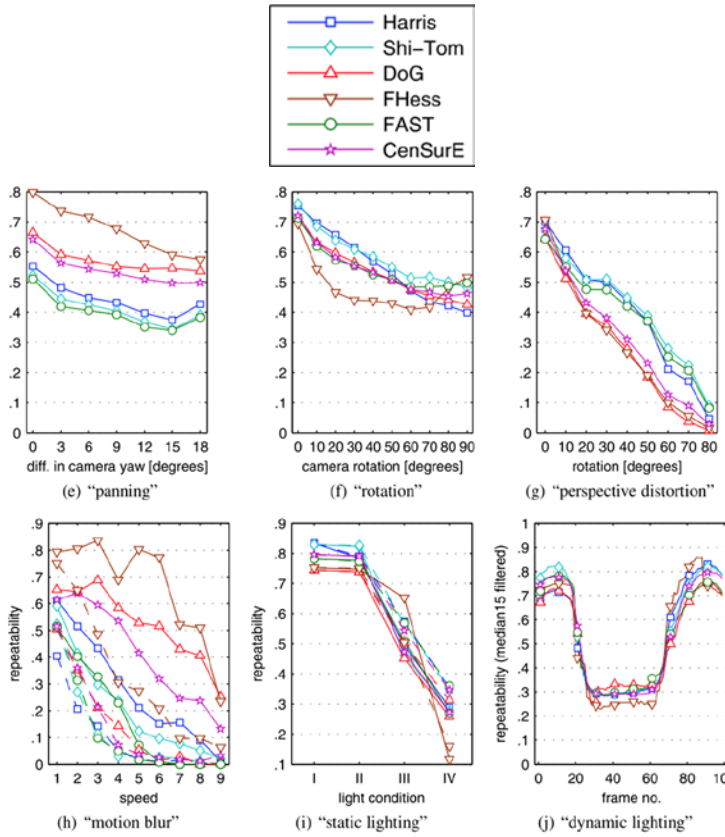


Figure 4-15. *Accuracy of feature descriptors over various invariance criteria. (From Gauglitz et al.[145], images © Springer Science +Business Media, LLC, used by permission)*

Turning to the more recent local binary descriptors, Alahi et. al. [130] provide a set of comparisons where FREAK is shown to be superior in accuracy to BRISK, SURF, and SIFT on a particular dataset and set of criteria developed by Mikolajczyk and Schmid [144] for feature accuracy over attributes such as viewpoint, blur, JPEG compression, brightness, rotation, and scale. In Rublee et. al. [120], ORB is shown to have better rotational invariance than SIFT, SURF, and BRIEF. In summary, local binary descriptors are proving to be attractive in terms of robustness, accuracy, and compute efficiency.

Accuracy Optimizations, Sub-Region Overlap, Gaussian Weighting, and Pooling

Various methods are employed to optimize feature descriptor accuracy, and a few methods are discussed here. For example, descriptors often use overlapping sampling pattern sub-regions, as shown in the FREAK descriptor pattern in Figure 4-9. By overlapping sampling regions and treating boundaries carefully, accuracy seems to be better in most all cases [161,178]. Overlapping regions makes sense intuitively, since each point in a region is related to surrounding points. The value of pattern sub-region overlapping in feature description seems obvious for local binary pattern type descriptors and spectra descriptor variants such as SURF and others [181,144]. When the sampling regions used in the descriptor do not overlap, recognition rates are not as accurate [130].

Gaussian weighting is another effective method for increasing accuracy to reduce noise and uncertainty in measurements. For example, the SIFT [161,178] descriptor applies a Gaussian-based weighting factor to each local area gradient in the descriptor region to favor gradients nearer the center and reduce the weighting of gradients farther away. In addition, the SIFT weighting is applied in a circularly symmetric pattern, which adds some rotational invariance; see Figure 6-17.

Note that Gaussian weighting is different from Gaussian filtering; a Gaussian filter both reduces noise and eliminates critical fine details in the image, but such filtering has been found to be counterproductive in the HOG method [106]. A Gaussian weighting factor, such as used by SIFT on the gradient bins, can simply be used to qualify data rather than change the data. In general, a weighting factor can be used to scale the results and fine-tune the detector or descriptor. The sub-region overlap in the sampling pattern and Gaussian weighting schemes are complementary.

Accuracy can be improved by relying on groups of nearby features together rather than just a single feature. For example, in convolutional networks, several nearby features may be pooled for a joint decision to increase accuracy via chosen robustness or invariance criteria [347]. The pooling concept may also be referred to as *neighborhood consensus* or *semi-local constraints* in the literature, and it can involve joint constraints, such as the angle and distance among a combined set of local features [348–350].

Sub-Pixel Accuracy

Some descriptor and recognition methods can provide sub-pixel accuracy in matching the feature location [147–151]. Common methods to compute sub-pixel accuracy include cross-correlation, sum-absolute difference, Gaussian fitting, Fourier methods, and rigid body transforms and ICP. In general, sub-pixel accuracy is not a common feature in popular, commercial applications and is needed only in high-end applications like industrial inspection, aerospace, and military systems.

For example, SIFT provides sub-pixel accuracy for the location of keypoints. Digital correlation methods and template matching are well known and used in industrial applications for object tracking, and can be extended to compute correlations over a range of one-pixel offset areas to yield a set of correlations that can be fit into a curve and interpolated to find the highest match to yield sub-pixel accuracy.

Sub-pixel accuracy is typically limited to translation. Rotation and scale are much more difficult to quantify in terms of sub-pixel accuracy. Typical sub-pixel accuracy results for translation only achieve better than ¼ pixel resolution, but resolution accuracy can be finer grained, and in some methods translation accuracy is claimed to be as high as $1/20^{th}$ of a pixel using FFT registration methods [151].

Also, stereo disparity methods benefit from improved sub-pixel accuracy, especially at long ranges, since the granularity of *Z* distance measurements increases exponentially with distance. Thus the calculated depth field contains coarser information as the depth field increases, and the computed depth field is actually nonlinear in *Z*. Therefore, sub-pixel accuracy in stereo and multi-view stereo disparity calculations is quite desirable and necessary for best accuracy.

Search Strategies and Optimizations

As shown in Figure 5-1, a feature may be sparse, covering a local area, or it may cover a regional or global area. The search strategy used to isolate each of these feature types is different. For a global feature, there is no search strategy: the entire frame is used as the feature. For a regional descriptor, a region needs to be chosen or segmented (discussed in Chapter 2). For sparse local features, the search strategy becomes important. Search strategies for sparse local regions fall into a few major categories, as follows (also included in the taxonomy in Chapter 5).

Dense Search

In a dense search, each pixel in the image is checked. For example, an interest point is calculated at each pixel, the interest points are then qualified and sorted into a candidate list, and a feature descriptor is calculated for each candidate. Dense search is used by local binary descriptors and common descriptors such as SIFT.

In stereo matching and depth sensing, each pixel is searched in a dense manner for calculating disparity and closest points. For example, stereo algorithms use a dense search for correspondence to compute disparity, line by line and pixel by pixel; monocular depth-sensing methods such as PTAM [327] use a dense search for interest points, followed by a sparse search for known features at predicted locations.

Dense methods may also be applied across an image pyramid, where the lower resolution pyramids are usually searched first and finer-grain pyramids are searched later. Dense methods in general are preferred for accuracy and robustness when feature locations are not known and cannot be predicted.

Grid Search

In grid search methods, the image is divided into a regular grid or tiles, and features are located based on the tiles. A novel grid search method is provided in the OpenCV library, using a grid search adapter (discussed in Chapter 6 and Appendix A). This allows for repeated trial searches within a grid region for the best features, and has the capability of adjusting detector parameters before each trial run. One possible disadvantage of a

grid search from the perspective of accuracy is that features do not line up into grids, so features can be missed or truncated along the grid boundary, decreasing accuracy and robustness overall.

Grid search can be used in many ways. For example, a regular grid is used as anchor points with the grid topology of D-NETS, as illustrated in Figure 4-7. Or, a grid is used to form image tile patches and a descriptor is computed for each tile, such as in the HOG method, as shown in Figure 4-12. Also the Viola Jones method [146] computes HAAR features on a grid.

Multi-Scale Pyramid Search

The idea behind the multi-scale image pyramid search is either to accelerate searching by starting at a lower resolution or to truly provide multi-scale images to allow for features to be found at appropriate scale. Methods to reduce image scale include pixel decimation, bilinear interpolation, and other multi-sampling methods. Scale space is a popular method for creating image pyramids, and many variations are discussed in the next section; see Figure 4-16.

Figure 4-16. *A five-octave scale pyramid. The image is from Albrecht Durer's Apocalypse woodcuts, 1498. Note that many methods use non-octave pyramid scales [120]*

However, the number of detected features falls off rapidly as the pyramid levels increase, especially for scale space pyramids, which have been Gaussian filtered, since Gaussian filters reduce image texture detail. Also, fewer pixels are present to begin with at higher pyramid levels, so a pyramid scale interval smaller than octaves is sometimes used. See reference[160] for a good discussion of image pyramids.

Scale Space and Image Pyramids

Often, instead of using simple pixel decimation and pixel interpolation to reduce image scale, a *scale space* [524,523] pyramid representation, originally proposed by Lindberg[547], is built up using Gaussian filtering methods to decrease the scaling artifacts and preserve blob-like features. Scale space is a more formal method of defining a multi-scale set of images, typically using a Gaussian kernel ***g()*** convolved with the image ***f(x,y),*** as follows:

$$g(x,y:t)=\frac{1}{2\pi t}e^{-(x^2+y^2)/2t}$$

$$L(.,.;t)=g(.,.:t)*f(.,.),$$

or by an equivalent method:

$$\partial_t L=\frac{1}{2}\nabla^2 L,$$

$$\text{with the initial state } L(x,y;0)=f(x,y),$$

A good example of Gaussian filter design for scale space is described in the SURF method [160]. Gaussian filters implemented as kernels with increasing size are applied to the original image at octave-spaced subsampling intervals to create the scale space images—for example, starting with a 9x9 Gaussian filter and increasing to 15x15, 21x21, 27x27, 33x33, and 39x39; see Figure 4-17.

Figure 4-17. *Scale space Gaussian images at scales of 0, 2, 4, 16, 32, 64. Image is from Albrecht Durer's Apocalypse woodcuts, 1498*

One drawback of scale space is the loss of localization and lack of accuracy in higher levels of the image pyramid. In fact, some features are simply missing from higher levels of the image pyramid, owing to a lack of resolution and to the Gaussian filtering. The best example of effective scale space feature matching may be SIFT, which provides for the 1st pyramid image in the scale to be double the original resolution to mitigate scale space problems, and also provides a good multi-scale descriptor framework. See also Figure 4-18.

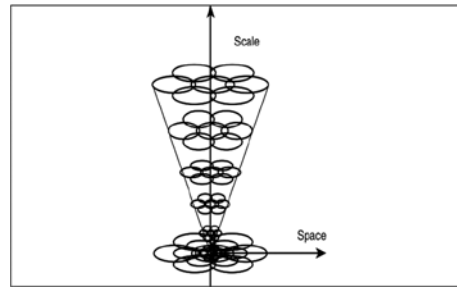


Figure 4-18. *Scale and space*

Image pyramids are analogous to texture *mip-maps* used in computer graphics. Variations on the image pyramid are common. Octave and non-octave pyramid spacings are used, with variations on the filtering method also. For example, the SIFT method [161,178] uses a five-level octave scale ***n/2*** image pyramid with Gaussians filtered images in a scale space. Then, the Difference of Gaussians (DoG) method is used to capture the interest point extrema maxima and minima in the adjacent images in the pyramid. SIFT uses a double-scale first pyramid level with linear interpolated pixels at 2x original magnification to help preserve fine details. This technique increases the number of stable keypoints by about four times, which is quite significant. In the ORB [120] method, a non-octave scale space is built around a scale over a five-level pyramid, which has closer resolution gradations between pyramid levels than an octave scale of two times.

Feature Pyramids

An alternative to scale space pyramids and pyramid searching is to use *feature-space pyramiding*, and build a set of multi-scale feature descriptors stored together in the database. In this approach, the descriptor itself contains the pyramid, and no scale space or image pyramid is needed. Instead, feature searching occurs directly from the mono-scale target image to the multi-scale features. The RFM method [220] discussed in Chapter 6 goes even further and includes *multi-perspective* transformed versions of each patch for each descriptor. In Table 4-3, note that the multi-scale features can be used to match directly on the target images, while the mono-scale features are better to use on an image pyramid.

Table 4-3. *Some Tradeoffs in Using a Mono-Scale Feature and a Multi-Scale Feature*

Feature Scale	Feature Size	Feature Description Compute Time	Image Pyramid Used for Matching	Mono-Scale Images Used for Matching
Mono-scale feature	Smaller memory footprint	Faster to compute	Yes	No
Multi-scale feature	Larger memory footprint	Slower to compute	No	Yes

Figure 3-16 shows the related concept of a *multi-resolution histogram* [152], created from image regions from a scale space pyramid and with the histograms concatenated in the descriptor that is used to determine texture metrics for feature matching. So in the multi-scale histogram method, no pyramid image set is required at run time; rather, the pyramid search uses histogram features from the descriptor itself to find correspondence with the mono-scale target image.

A wide range of scalar and other metrics can be composed into a multi-scale feature pyramid, such as image intensity patches, color channel intensity patches, gradient magnitude, and gradient orientations. Histograms of textural features have been found useful as affine-invariant metrics as a part of a wider feature descriptor [152].

Sparse Predictive Search and Tracking

In a sparse predictive search pipeline, specific features at known locations, found in previous frames, are searched for in the next frame at the expected positions. For example, in the PTAM [327] algorithm for monocular depth sensing, a sparse 3D point cloud and camera pose are created from sequential video frames from a single camera by locating a set of interest points and feature descriptors. For each new frame, a *prediction* is made of the coordinates where the same interest points and feature detectors might be in the new image, using the prior camera pose matrix. Then, for the new frame, a search or tracking loop is started to locate a *small number* of the predicted interest points using a pyramid coarse to fine search strategy. The predicted interest points and features are searched for within a range around where each is predicted to be, and the camera pose matrix is updated based on the new coordinates where the features are found. Then, a *larger number* of points are predicted using the updated camera pose and a search and tracking loop is entered over a finer scale pyramid image in the set. This process iterates to find points and refine the pose matrix.

Tracking Region-Limited Search

One example of a region-limited search is a video conferencing system that tracks the location of the speaker using stereo microphones to calculate the coarse location via triangulation. Once the coarse speaker position is known, the camera is moved

to view the speaker, and only the face region is of interest for further fine positional location adjustments, auto-zoom, auto-focus, and auto-contrast enhancements. In this application, the entire image does not need to be searched or processed for face features. Instead, the center of the FOV is the region where the search is limited to locate the face. For example, if the image is taken from an HD camera with 1920x1080 resolution, only a limited region in the center of the image, perhaps 512x512 pixels, needs to be processed to locate the face features.

Segmentation Limited Search

A segmented region can define the search area, such as a region with specific texture, or pixels of a specific color intensity. In a morphological vision pipeline, regions may be segmented in a variety of ways, such as thresholding and binary erosion + dilation to create binary shapes. Then the binary shapes can be used as masks to segment the corresponding gray scale image regions under the masks for feature searching. Image segmentation methods were covered in Chapter 2.

Depth or *Z* Limited Search

With the advent of low-cost commercial depth sensors appearing on mobile consumer devices, the *Z* dimension is available for limiting search ranges. See Figure 4-19. For example, by segmenting out the background of an image using depth, the foreground features are more easily segmented and identified, and search can be limited by depth segments. Considering how much time is spent in computer vision to extract 3D image information from 2D images, we can expect depth cameras to be used in novel ways to simplify computer vision algorithms.

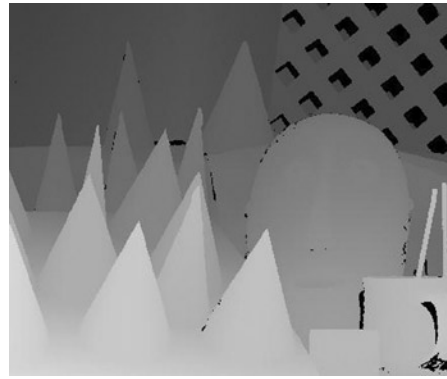


Figure 4-19. *Segmentation of image regions based on a depth map. Depth image from Middlebury Data set: (Source: D. Scharstein and C. Pal "Learning conditional random fields for stereo" CVPR Conference, 2007. Courtesy of authors)*

Computer Vision, Models, Organization

This section contains a high- level overview of selected examples to illustrate how feature metrics are used within computer vision systems. Here, we explore how features are selected, learned, associated together to describe real objects, classified for efficient searching and matching, and used in computer vision pipelines. This section introduces machine learning, but only at a high level using selected examples. A good reference on machine learning is found in [546] by Prince. A good reference for computer vision models, organization, applications, and algorithms is found in Szelinski [324].

Several terms are chosen and defined in this section for the discussion of computer vision models, namely *feature space*, *object models*, and *constraints*. The main topics for this section include:

- Feature spaces and selection of optimal features
- Object recognition via object models containing features and constraints
- Classification and clustering methods to optimize pattern matching
- Training and learning

■ **Note** Many of the methods discussed in computer vision research journals and courses are borrowed from other tangent fields and applied, for example, machine learning and statistical analysis. In some cases computer vision is driving the research in such tangent fields. Since these fields are well established and considered beyond the scope of this work, we provide only a brief topical introduction here, with references for completeness [546,324].

Feature Space

The collection and organization of all features, attributes, and other information necessary to describe objects may be called the *feature space*. Features are typically organized and classified into a feature space during a training or learning phase using ground truth data as a training set. The selected features are organized and structured in a database or a set of data structures, such as trees and lists, to allow for rapid search and feature matching at run time.

The feature space may contain one or more types of *descriptors* using spectra such as histograms, binary pattern vectors, as multivariate composite descriptors. In addition, the feature space contains *constraints* used to associate sets of features together to identify objects and classes of objects. A feature space is unique to any given application, and is built according to the types of features used and the requirements of the application; there is no standard method.

The feature space may contain several *parameters* for describing objects; for example:

- **Several types of feature descriptors**, such as SIFT and simple color histograms.
- **Cartesian coordinates** for each descriptor relative to training images.
- **Orientations** of each descriptor.
- **Name of training image** associated with each descriptor.
- **Multimodal information**, such as GPS, temperatures, elevation, acceleration.
- **Feature sets** or lists of associated descriptors.
- **Constraints** between the descriptors in a set, such as the relative distance from each other, relative distance thresholds, angular relationships between descriptors, or relative to a reference point.
- **Object models** to collect and associate parameters for each object.
- **Classes** or associations of objects of the same type, such as automobiles.
- **Labels** for objects or constraints.

Object Models

An *object model* describes real objects or classes of objects using parameters from the feature space. For example, an object may contain all parameters required to describe a specific automobile, such as feature descriptor sets, labels, and constraints. A class of objects may associate and label all objects of the same class, such as an automobile of any type. There is no standard or canonical object model to follow, so in this section we describe the overall attributes of computer vision objects and how to model them.

Object models may be composed of sets of individual features; constraints on the related features, such as position or orientation of features within an object model; and perhaps other multimodal information for the objects or descriptors, such as GPS information or time stamps, as shown in Figure 4-20. The object model can be created using a combination of supervised and unsupervised learning methods [403]; we survey several methods later in this chapter.

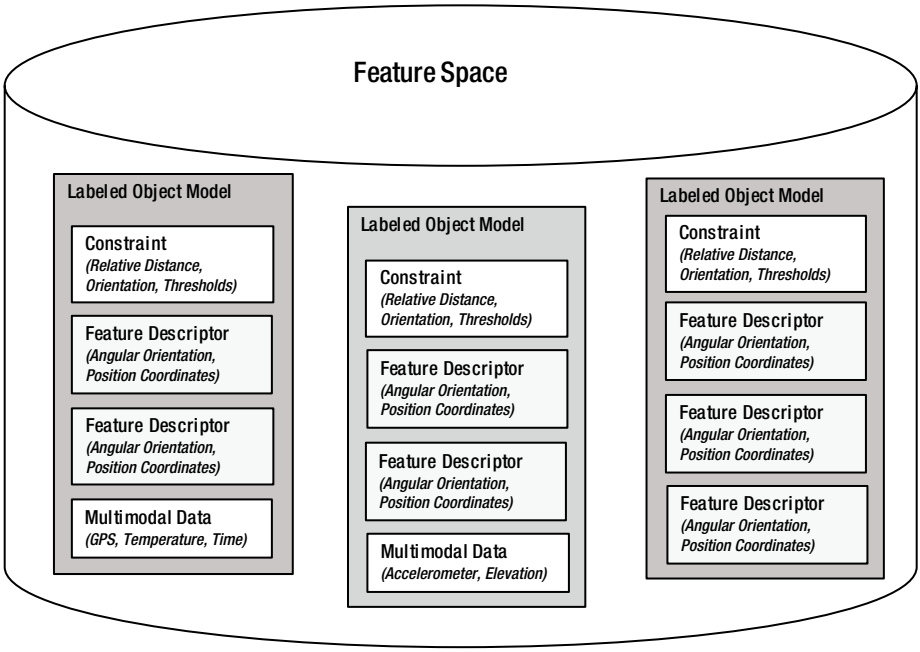


Figure 4-20. *Simplified hypothetical feature space showing organization and association of features, constraints, and objects*

One early attempt to formulate object models is known as *parts-based models,* suggested in 1973 by Fischler and Elschlager[530]. These describe and recognize larger objects by first recognizing their parts—for example, a face being composed of parts such as eyes, nose, and mouth. There are several variations on parts-based models; see references[531–533], for example. Machine learning methods are also used to create the object models [546], and are discussed later in this section.

A simple object model may be composed of only image histograms of whole images, the name or label of each associated image, and possibly a few classification parameters such as the subject matter of the image, GPS location, and date. To identify unknown target images, a histogram of the target image is taken and compared against image histograms from the database. Correspondence is measured using a suitable distance metric such as SAD. In this simple example, brute-force searching or a hash table index may be used to check each histogram in the database against target image histograms, and perhaps other parameters from the object model may be matched along with the histograms, such as the GPS coordinates. No complex machine learning classification, clustering, data reductions, or organization of the database need be done, since the search method is brute-force. However, finding correspondence will become progressively slower as more images are added to the database. And the histogram all by itself is not very discriminative and offers little invariance.

Constraints

Key to object recognition, *constraints* are used to associate and qualify features and related attributes as objects. Features alone are probably insufficient to recognize an object without additional qualification, including *neighborhood consensus* or *semi-local constraints* involving joint constraints, such as the angle and distance among a combined set of local features [348–350]. Constraints associate object model elements together to describe and recognize a larger object [365,366,379], such as by minimum feature count thresholds required to ensure that a proper subset of object features are found together, or by using multimodal data constraints such as GPS position, or by voting.

Since there are many approaches for creating constraints, we can only illustrate the concept. For example, Lowe[161] shows recognition examples illustrating how SIFT features can be used to recognize objects containing many tens of distinct features, in some cases using as few as two or three good features. This allows for perspective and occlusion invariance if some of the features describing the object cannot be found, taking into consideration feature orientation and scale as constraints. Another example is wide baseline stereo matching, which requires position and distance constraints on feature pairs in L/R image assuming that the scale and orientation of L/R feature pairs is about equal; in this case, translation would be constrained to be within a range based on depth.

Selection of Detectors and Features

Feature detectors are selected based on a combination of variables, such as the feature detector *design method* and the types of invariance and performance desired. Several approaches or design methods are discussed next.

Manually Designed Feature Detectors

Some feature detectors, such as polygon shape descriptors and sparse local features like SURF, are manually designed and chosen using the intuition, experience, and test results of the practitioner to address the desired invariance attributes for an application. This involves selecting the right spectra to describe the features, determining the shape and pattern of the feature, and choosing the types of regions to search. However, some detectors are statistically and empirically designed, which we cover next.

Statistically Designed Feature Detectors

Statistical methods are used to design and create feature detectors. For example, the binary sampling patterns used in methods such as ORB and FREAK are created from the training dataset based on the statistical characteristics of the possible interest point comparison pairs. Typically, ORB ranks each detected interest point feature pair combination to find terms that are uncorrelated with high variance. This is a statistical sorting or training process to design the feature patterns and tune them for a specific ground truth dataset. See Figure 4-11 for more details on ORB, and see the discussions of FREAK and ORB earlier in this chapter as well.

SIFT also uses statistical methods to determine, from a training set, the best interest points, dominant orientation of each interest point, and scale of each interest point.

Learned Features

Many systems learn a unique codebook of features, using sparse coding methods to identify a unique set of basis features during a training phase against selected ground truth data. The learned basis features are specific to the application domain or training data, and the chosen detectors and descriptors may simply be pixel regions used as correlation templates. However, any descriptor may be used, such as SIFT. Neural network and convolutional network approaches are popularly used for feature learning, as well as sparse coding methods, which are discussed later in this chapter.

Overview of Training

A machine vision system is *trained* to recognize desired features, objects, and activities. However, training can be quite complex and is covered very well in the field of machine learning and statistical analysis (which we do not cover in any detail). Training may be supervised and assisted by an expert, or unsupervised as in the deep learning methods discussed later in this section. Here, we provide an overview of common steps and provide references for more detail. One of the simplest examples of training would be to take image histograms associated with each type of image—for example, a set of histograms that describe a face, animal, or automobile taken from different images.

Training involves collecting a training set of images appropriate for the application domain, and then determining which detectors and descriptors can be tuned to yield the best results. In some cases, the feature descriptor itself may be trainable and designed to match the training data, such as the local binary pattern descriptors ORB, BRIEF, and FREAK, which can use variable pixel sampling patterns optimized and learned from the training data.

In feature learning systems, the entire feature set is learned from the training set. Feature learning methods employ a range of descriptor methods such as simple correlation temples containing pixel regions, or SIFT descriptors. The learned feature set is reduced by keeping only the features that are significantly different from features already in the set. Feature learning methods are covered later in this chapter.

To form larger objects during training, sets of features are associated together using constraints, such as geometric relationships like angles or distances between features, or the count of features of a given value within a specific region. Objects are determined during training, which involves running detectors and descriptors against chosen ground truth data to find the features, and then determining the constraints to represent objects as a composite set of features. Activities can be recognized by tracking features and their positions within adjacent frames, so activity can be considered a type of meta-object and stored in a database as well.

In any case, the features obtained through the training phase are *classified* into a searchable feature space using a wide range of statistical and machine learning methods. Training, classification, and learning are discussed at a high level later in this chapter.

Classification of Features and Objects

Classification is another term for recognition, and it includes feature space organization and training. A *classifier* is a term describing a method or system for learning structure from data and recognizing objects. Several approaches are taken for automatically building classifiers, including support vector machines (SVMs), kernel machines, and neural networks.

In general, the size of the training set or ground truth dataset is key to classifier accuracy [336–338]. During system training, first a training set with ground truth data is used to build up the classifier. The machine learning community provides a wealth of guidance on training, so we defer to established sources. Key journals to dig deeper into machine learning and testing against ground truth data include NIPS and IEEE PAMI, the latter which goes back to 1979. Machine learning and statistical methods are used to guide the selection, classification, and organization of features during training. If no classification of the feature space is made, the feature match process follows a slow brute-force linear search of new features against known features.

Key classification problems discussed in this section include:

- **Group Distance and Clustering** of similar features using a range of nearest–neighbor methods to assist in organization, fitting, error minimization, searching and matching, and enabling similarity constraints such as geometric proximity, angular relationships, and multimodal cues.
- **Dimensionality Reductions** to avoid over-fitting, cleaning the data to remove outliers and spurious data, and reducing the size of the database.
- **Boosting and Weighting** to increase the accuracy of feature matching.
- **Constraints** describing relationships between descriptors composing an object, such as pose estimators and threshold accept/reject filters.
- **Structuring the Database** for rapid matching vs. brute-force methods.

Group Distance: Clustering, Training, and Statistical Learning

We refer to *group distance* and *clustering* in this discussion, sometimes interchangeably, as methods to describe similarities and differences between groups of data atoms, such as feature descriptors. Applications of group distance and clustering include error minimization, regression, outlier removal, classification, training, and feature matching.

According to Estivill-Castro[351], *clustering* is impossible to define in a mathematical sense, since there are so many diverse methods and approaches to describe a cluster. See Table 4-3 for a summary of related methods. However, we will discuss clustering here in the context of computer vision to address data organization, pattern matching, and describing object model constraints (while attempting to not ruffle the feathers of mathematical purists who use different terminology).

To identify similar features in a group, a wide range of clustering algorithms or group distance algorithms are used [353], which may also be referred to as *error minimization* and *regression methods* in some literature. Features are clustered together for computer vision to help solve fundamental problems, including object modeling, finding similar patterns during matching, organizing and classifying similar data, and dimensionality reductions.

One way to describe a cluster is by *similarity*—for example, describing a cluster of related features under some distance metric or regression method. In this sense, clustering overlaps with distance functions: Euclidean distance for position, cosine distance for orientation, and Hamming distance for binary feature vector comparisons are examples. However, distance functions between two points are differentiated in this discussion from group distance functions, clusters, and group distributions.

Efficiently organizing similar data in feature space for searching and classification is a form of clustering. It can be based on similarity or distance measures of feature vectors or on object constraint similarity, and it is required to speed up feature searching and matching. However, commercial databases "and brute-force search" may be used as-is for feature descriptors, with no attempt made to optimize. Custom data structures can be built for optimizations via trees, pyramids, lists, and hash tables. (We refer the reader to standard references in computer science covering data organization and searching; see the classic texts *The Art of Computer Programming* by Donald Knuth or *Data structure and Algorithms* by Aho, Ullman, and Hopcroft.)

Another aspect of clustering is the feature space dimension and topology. Since some feature spaces are multivariate and multidimensional, containing scalars and tensors, any strict definition of clustering, error minimization, regression, or distance is difficult; it really depends on the space in which similarity is to be measured.

Group Distance: Clustering Methods Survey, KNN, RANSAC, K-Means, GMM, SVM, Others

A spectrum of alternatives may be chosen for clustering and learning similarities between groups of data atoms, starting at the low end with basic C library searching and sorting functions, and reaching the high end with statistical and machine learning methods such as kernel machines and support vector machines (SVMs) to build complete classifiers. Kernel machines allow various similarity functions to be substituted into a common framework to enable simplified comparison of similarity methods and classification.

Table 4-4 is a summary of selected clustering methods, with a few key references for the interested reader.

Table 4-4. *Clustering, Classification, and Machine Learning Methods*

Group Distance Criteria	Methods & References	Description
Distance	K-Nearest Neighbor [364]	Uses a chosen distance function, cluster based on simple distance to k -nearest neighbors in the training set.
Consensus Models	RANSAC [380] PROSAC [363] Levenberg-Marquardt [401]	Use random sample consensus to estimate model parameters from contaminated data sets.
Centroid Models	K-Means [354], Voroni Tesselation, Delauney Triangulation Hierarchical K-Means, Nister trees [387]	Use a centroid of distribution as the base of the cluster, which can be very slow for large datsasets; can be formulated in a hierarchical tree structure using vocabulary words (Nister method) for much better performance.
Connectivity of Clusters	Hierarchical Clustering [355]	Builds connectivity between other clusters.
Density Models	DBSCAN [395][352] OPTICS [396]	Locate distributions with maxima and minima density compared to surrounding data.
Distribution Models	Gaussian Mixture Models [356]	Iterative methods of finding maximum likelihood of model parameters.
Neural Methods	Neural Networks [360]	Neural methods defy a single definition, but typically use one or more inputs; adaptive weight factors for each input that can be learned and trained, a neural function to act on the inputs and weights, a bias factor for the neural function; produce one or more outputs.
Bayesian	Naïve Bayesian [383] Randomize Trees [384] FERNS [307]	Learning model recording probabilistic relationships between variables.

(*continued*)

Table 4-4. (*continued*)

Group Distance Criteria	Methods & References	Description
Probabilistic, Semantic	[232] Latent Semantic Analysis (pLSA) Latent Dirichlet Allocation (LDA) Hidden Markov Models, HMM [385][386]	Learning model based on probabilistic relationships between variables.
Kernel Methods, Kernel Machines	Kernel Machines [361][1] Various Kernels [362] PCA [357][358] *SVM is a well-known instance of a kernel machine.	Reduce a distribution to a set of uncorrelated, ranked principal components in a Euclidean space for ease of matching and clustering.
Support Vector Machines	SVM [377,359]	An SVM may produce structured or multivariate output to classify input.

[1] http://www.kernel-machines.org/

Classification Frameworks, REIN, MOPED

Training and classification fall into the following general categories:

- **Supervised.** A human will assist during the training process to make sure the results are correct.
- **Unsupervised.** The classifier can be trained automatically from feature data and parameters [403].

Putting all the pieces together, we see that training the classifiers may be manual or automated, simple or complex, depending on the complexity of the objects and the range of feature metrics used.

An SVM or kernel machine may be the ideal solution, or the problem may be simpler. For example, a machine vision system to identify fruit may contain a classifier for each type of fruit, with features including simple color histograms, shape factors such as area and perimeter and Fourier descriptors, and surface texture metrics, with constraints to associate and quantify all the features for each type of fruit. The training process would involve imaging several pieces of fruit of each type; developing canonical descriptors for color, shape, and surface texture; and devising a top-level classifier perhaps discriminating first on color, next surface texture, and finally shape. A simpler fruit classifier may contain just a set of image histograms of accurate color measurements for each fruit object, and may work well enough if each piece of fruit is imaged with a high-precision color camera against a black conveyor belt background in a factory.

While most published research is based on a wide range of nonstandard classification methods designed for specific applications or to demonstrate research results, some work is being done toward more standardized classification frameworks.

One noteworthy example of a potentially standard classifier framework developed for robot navigation and object recognition is the REIN method [397], which allows the mixing and matching of detectors, descriptors, and classifiers for determining constraints. REIN provides a plug-in architecture and interfaces to allow for any algorithms, such as OpenCV detectors and descriptors, to be combined in parallel or serial pipelines. Two classification methods are available in REIN as plug-in modules for concurrent use: *Binarized Gradient Grid Pyramids* are introduced as a new method [397], and *View Point Feature Histograms* [398] are also used.

The REIN pipeline provides interfaces for (1) *attention operators* to identify interesting 3D points and reduce the search space; (2) *detectors* for creating feature descriptors; and (3) *pose estimators* to determine geometric constraints for applications like robot motion such as grasping. REIN is available for research as open source; see reference[397].

Another research project, MOPED [399], provides a regular architecture for robotic navigation, including object and pose recognition. MOPED includes optimizations to use all available CPU and GPU compute resources in parallel. Moped provides optimized versions of SIFT and SURF for GPGPU, and makes heavy use of SSE instructions for pose estimation.

Kernel Machines

In machine learning, a *kernel machine* [362] is a framework allowing a set of methods for statistically clustering, ranking, correlating, and classifying patterns or features to be automated. One common example of a kernel machine is the support vector machine (SVM) [341].

The framework for a kernel machine maps descriptor data into a feature space, where each coordinate in the feature space corresponds to a descriptor. Within the feature space, feature matching and feature space reductions can be efficiently carried out using *kernel functions*. Various kernel functions are used within the kernel machine framework, including RBF kernels, Fisher kernels, various polynomial kernels, and graph kernels.

Once the feature descriptors are transformed into the feature space, comparisons, reductions, and clustering may be employed. The key advantage of a kernel machine is that the kernel methods are interchangeable, allowing for many different kernels to be evaluated against the same feature data. There is an active kernel machine community (see `kernel-machines.org`).

Boosting, Weighting

Boosting [381] is a machine learning concept that allows a set of classifiers to be used together, organized into combinatorial networks, pipelines, or cascades, and with learned weights applied to each classifier. This results in a higher, synergistic prediction and recognition capability using the combined weighted classifiers. Boosting is analogous to the weighting factors used for neural network inputs; however, boosting methods go further to combine networks of classifiers to create a single, strong classifier.

We will illustrate boosting from the Viola Jones method [146,186] also discussed in Chapter 6, which uses the ADA-BOOST training method to create a cascaded pattern matching and classification network by generating strong classifiers from many weak learners. This is done through dynamic weighting factors determined in a training phase, and the method of using weighting factors is called *boosting*.

The idea of boosting is to first start out by equally weighting the detected features—in this case, HAAR wavelets—and then matching the detected features against the set of expected features; for example, those features detected for a specific face. Each set of weighted features is a classifier. Classifiers that fail to match correctly are called *weak learners*. For each weak learner during the training phase, new weighting factors are applied to each feature to make the classifier match correctly. Finally, all weak learners are combined linearly into a *cascaded classifier,* which is like a pipeline or funnel of weak classifiers designed to reject bad features early in the pipeline.

The training can take many hours, days or weeks and requires some supervision. While ADA-BOOST solved binary classification problems, the method can be extended into multiclass classification [382].

Selected Examples of Classification

We call out a few noteworthy and popular classification approaches here, which are also listed in Table 4-5.

Table 4-5. *Comparison of Various Interest Point, Descriptor, and Classifier Concepts*

Technique	FERNS	SIFT	FREAK	Convolutional Network	Polygon Shape Factors
Sparse Keypoints	x	x	x	x	
Feature Descriptor		x	x	x	x
Multi-Scale Representation	x	x		x	
Coarse to Fine Descriptor			x		
Deep Learning Network				x	
Sparse Codebook				x	

Note: The FERNS method does not rely on a local feature descriptor, and instead relies on a classifier using constraints between interest points.

Randomized trees is a method using hierarchical patch classifiers [384] based on Bayesian probability methods, taking a set of simple patch features deformed by random homography parameters. Ozuysal et al.[307] further develop the randomized tree method with optimizations using non-hierarchical organization in the form of *FERNS,* using binary probability tests for patch classifier membership. Matches are evaluated using a naïve Bayesian approach.

FERNS training [307] involves combining training data from multiple viewpoints of each patch to add scale and perspective invariance, using trees with 11 levels and 11 versions of each patch, warped using randomized affine deformation parameters; some Gaussian noise and smoothing are also applied to the deformed patches. Keypoints are then located in each deformed patch, and the keypoints found in the most deformed patches are selected for the training set. The FERNS keypoints use maxima of Laplacian filters at three scales and retain only the strongest 400 keypoints. The Laplacian keypoints do not include orientation or fine-scale estimation. FERNS does not use descriptors, just the strongest Laplacian keypoints computed over the 11 deformed images in each set.

While K-means [354] methods can be very slow, an optimization using hierarchical Nister Trees [387] is a highly scalable alternative for indexing massive numbers of quantized or clustered local descriptors in a hierarchical vocabulary tree. The method is reported to be very discriminative and has been tested on large datasets.

Binary Histogram Intersection Minimization (BHIM) [322] uses pairs of multi-scale local binary patterns (MSLBP) [322] to form pairwise-coupled classifiers based on strong divergence between pairs of MSLBP features. Histogram intersection on pairs of MSLBP features use a distance function such as SAD to find the largest divergence of histogram distance. The BHIM classifier is then composed of a list of "pairs" of MSLBP histograms with large divergence, and MSLBPs are matched into the classifier. BHIM uses features created across multiple scales of training data. It is reported by the authors to be at least as accurate as ADA-BOOST, and the MSLBP features are reported to be more discriminant than LBPs.

Alahi et al.[391] develop a method for classification and matching using a cascaded set of coarse to fine grids of region descriptors called *object descriptors* (ODs). The target application is tracking objects across a set of cameras, such as traffic cameras in a metropolitan area. Each OD is a collection of multi-scale descriptors computed in equal-size regions over multi-scale grids; the grids range over six scales with a 25 percent scaling factor difference. Any existing descriptor method can be used in the OD method, such as SIFT, SURF, or correlation templates. The authors [391] claim improved performance by cascading descriptors in an OD compared with using existing descriptors.

Feature Learning, Sparse Coding, Convolutional Networks

Feature learning methods create a set of basis features (we use the term *basis features* loosely here) derived from the ground truth data during a training phase. The basis features are collected into a set. There are several related approaches taken to create the set, discussed in this section.

Terminology: Codebooks, Visual Vocabulary, Bag of Words, Bag of Features

Several related approaches and terminologies are used in the feature learning literature, including variations such as *sparse coding*, *codebooks*, *bag of words*, and *visual vocabularies*. However, for the novice, there is some conceptual overlap in the various

approaches and the terminology is subtle, describing minor variations in methods used to learn the features and build the classification networks; see references[114–119]. The sparse codes are analogous to basis features. Many researchers in the areas of activity recognition [69,75] are using sparse codebooks and extending the field of research.

We describe some of the terminology and concepts, including:

- Dictionaries, codebooks, visual vocabularies, bags of words, bags of features, and feature alphabet, containing sets of features.
- Sparse codes, sparse coding, and minimal sets of features or codes.
- Multi-layered sparse coding and deep belief networks, containing multi-layered classification networks for hierarchical matching; these are composed of small, medium, and large scale features—perhaps ten or more layers of scale.
- Single-layer sparse coding, with no hierarchy of features, which may be built on top of a multi-scale descriptor such as SIFT.
- Unsupervised feature learning, including various methods of learning the best features for a given application from the ground truth dataset; feature learning has received much attention recently in the Neural Information Processing Systems (NIPS) community, especially as applied to convolutional networks.

Sparse Coding

Some early work in the area of sparse coding for natural images can be found in the work of Olshausen and Field [126], which forms the conceptual basis. To create a *sparse codebook*, first an image feature domain is chosen, such as face recognition or automobile recognition. Then a set of *basis items* (patches, vectors, or functions) are selected and put into a codebook based on a chosen uniqueness function. The sparse coding goal is to contain the smallest set of unique basis items required to achieve the accuracy and performance goals for the system.

When adding a new feature to the codebook during the training stage, candidate features are compared against the features already in the codebook to determine feature uniqueness, using a suitable distance function and empirical threshold. If the feature is sufficiently unique, as measured by the distance function and a threshold, the new feature is added to the codebook.

In work by Bo, Ren, and Fox[124], the training phase involves using objects such as a cup, which is positioned on a small rotating table. Multiple images are taken of the object from a number of viewpoints and distances to achieve perspective invariance, which then yields a set of patches taken from a variety of poses, from which the unique sparse codewords are created and added to the codebook. See also references[124,237,225,226]. Related work includes a histogram of sparse codes descriptor or HSC [125], as described in Chapter 7, used to retrofit a HOG descriptor.

Visual Vocabularies

Visual vocabularies are analogous to word vocabularies and they share common research [231]. In the area of document analysis, content is analyzed and described based on the histogram of unique word counts in the document. Of course, the histogram can be trimmed and remapped to reduce the quantization and binning. Visual vocabularies follow the same method as word vocabulary methods, representing images globally by the frequency of visual words, as illustrated in Figure 4-21, where visual word methods use feature descriptors of many types.

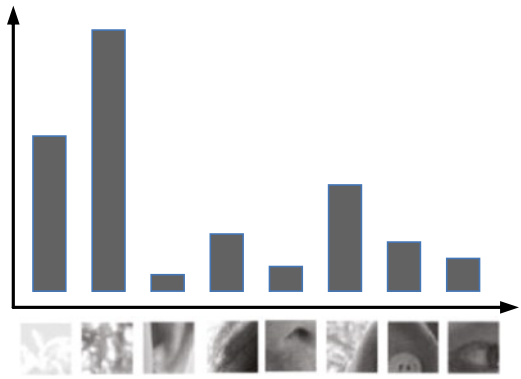


Figure 4-21. *Hypothetical, simplified illustration representing a set of visual words, and a histogram showing frequency of use of each visual word in a given image*

To build the visual vocabularies, unique features descriptors are extracted and collected from ground truth images. To be included in the vocabulary, the new feature must have significant statistical differences from the existing features in the vocabulary, so features are added to the vocabulary only if they exceed a difference threshold function.

To quantize the visual vocabulary features for determining their uniqueness, clustering and classification methods are performed on the feature set, and candidate features are selected that are unique so as to reduce the feature space and assist in matching speed. Various statistical methods may be employed to reduce the feature space, such as K-means, KNN, SVM, Bayes, and others.

To collect the visual features, practitioners are using all possible methods of feature description and image search, including sampling the image at regular grids and at interest points, as well as scale space searches. The features used in the vocabularies range from simple rectangular pixel regions, to SIFT features, and everything in between. Applications for the visual vocabularies range from analyzing spatio-temporal images for activity recognition [232,235] to image classification [233,234,118,116,235].

Learned Detectors via Convolutional Filter Masks

As illustrated in Figure 4-22, Richardson and Olson[122] developed a method of learning optimal convolutional filters as an interest point detector with applications to stereo visual odometry. This method uses combinations of DCT and HAAR basis features composed together, using random weights to form a set of candidate 8x8 pixel basis functions, each of which is tested against a target feature set resembling 2D barcodes known as AprilTags [527]. Each 8x8 pixel candidate is measure against the AprilTags to find the best convolution masks for each tag to form the basis set. Of course, other target features such as corners could be used for ground truth data instead of AprilTags.

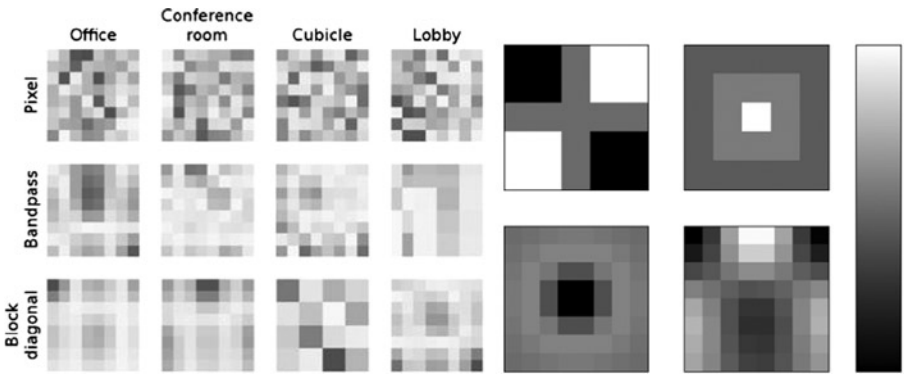


Figure 4-22. *(Left) The optimal learned convolution filters for an image of an Office, a conference room, cubicle, and lobby; gray scale values represent filter coefficient magnitudes. (Right) Comparable corner detectors in the top row, difference of Gaussian in the bottom left, and a custom filter which is preferred by the author. (Images © Andrew Richardson and Edwin Olson, used by permission)*

Using the learned convolution masks, the steps in feature detection are as follows: (1) convolve each masks at chosen pixels to get a response; (2) compare convolution response against a threshold; (3) suppress non-extrema response values using a 3x3 spatial filter window. The authors report good accuracy and high performance on the order of a FAST detector, but with the benefit of higher performance for the combined detection and non-maximal suppression stage as feature counts increase.

Convolutional Neural Networks, Neural Networks

Convolutional neural networks, pioneered by Lecun [339] and others, are one method of implementing machine learning algorithms based on neural network theory [360]. Convolutional networks are showing great success in academia and industry [340] for image classification and feature matching.

Convolutional neural networks are one method of modeling a neural network. The main compute elements in the convolutional network are many optimized convolutions in parallel, as well as fast local memory between the compute units. The run-time classification performance can be quite fast, especially for hardware-optimized implementations [528].

As shown in Figure 4-23 at a high level, one method of modeling each neuron and a network of neurons includes a set of inputs, a set of weighting factors applied to each input, a combinatorial function, and an output. Many neural models exist that map into convolutional networks, we refer the reader to the experts, see Lecun [339]. Neural networks have been devised using several models, but this topic is outside the scope of this work [360]; see the NIPS community research for more.

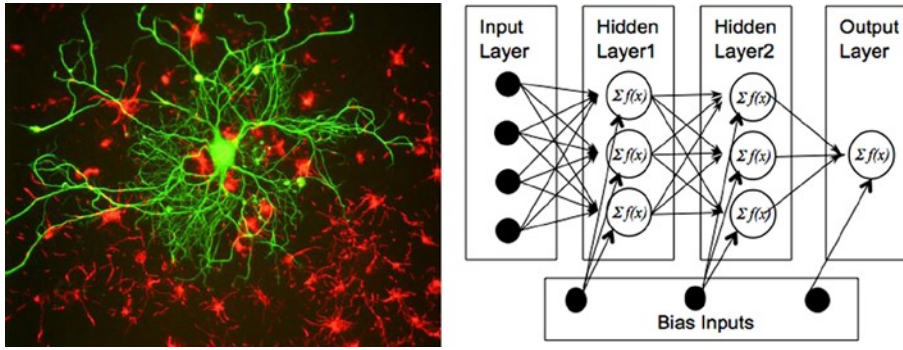


Figure 4-23. *(Left) Neurons from a human brain. (Right) One of many possible models of an artificial neural network [360]. Note that each neuron may have several inputs, several outputs, a bias factor, and input/output weight factors (not shown). Human neuron Image on left @ Gerry Shaw, used by permission*

Neural networks are multi-level, containing several layers and interconnections. As shown in the hypothetical neural network in Figure 4-23, a bias input is provided to each neural function as a weighting factor. Some neural network configurations use individual weights applied to each individual input, so the weighting factors act as convolution kernel coefficients. In terms of convolutional networks, the neural network paradigm can be mapped into localized patches of raw pixels as feature inputs at the lowest level. For example, the patch size may be 1 pixel or a 5x5 patch of pixels, each input having a convolutional weighting factor.

Learned weighting factors [85,339] are determined in the networks to use as convolution kernel values applied to each pixel in the patch. The output of a layer is referred to as a *feature map*. The weighting factors are learned in the network, and may be back-propagated to tune the system during training.

A standard introduction to convolutional networks is provided by Lecun [339]. During the learning process, a key goal is to preserve only the unique features and reduce the feature space; for this reason, sparse coding is used. Learned features are composed into a multi-layer structure of scaled high-level, mid-level, and low-level features in a *deep learning* approach [339,340] containing 10 or more scale layers. Networks and pixel input areas may overlap into adjacent convolutional kernels.

Deep Learning, Pooling, Trainable Feature Hierarchies

Local feature descriptors are often concerned with matching at a specific scale or perhaps even a few scales. However, trainable feature hierarchy methods [402,339] are being developed that classify features using a hierarchy, or *deep set,* of features containing low-level features at fine scales, intermediate, or medium scale features, and high-level features at coarse scales—perhaps eight or more layers in the feature detection hierarchy—producing deeper representations, which is the goal of deep learning AI methods [525].

A deep learning approach may include several layers of neural networks, including hidden layers. To reduce the feature space at each level of the hierarchy, feature learning is used at each level to pool [404] similar local features, preserving only the unique features. Various methods of feature pre-processing are used for pooling, such as feature whitening [405], to normalize features to be similar under contrast or variance. The low-level features may include local region pixel details, and the high-level features may be similar to regional shape metrics. Such trainable feature classification networks are discussed in the literature under many names, such as deep belief networks [526] and feature learning.

Many researchers are building deep belief networks relying on rectangular pixel patches for the feature, and are using convolution or correlation for the feature matching method. Convolutional networks using deep learning are deployed in many successful commercial applications, such as speech recognition, or face, person, and gender recognition. They have also been used to win several competitions [340]. Convolutional networks using deep learning are reported to increase in accuracy as the resolution of features decreases toward a finer scale, which increases the depth of the network. Training is reported to take several days [340], using a bank of dedicated GPUs.

One interesting example is the work of Bo, Ren and Fox [242], where a hierarchical matching pursuit HMP method (deep method) is employed to learn features in an unsupervised framework and add to a sparse codebook with two levels. RGB-D data channels are used to compute the descriptors, including separate descriptors for gray scale or intensity, *RBG* color, *Z* or depth from a depth camera, and the 3D surface normal from the depth data. A few different descriptor sizes are used, including 16x16 patches sampled with 4-pixel overlap for higher-level matching, and a set of nonoverlapping 5x5 patches for lower levels. The features are pooled as a part of the feature learning process.

Summary

In this chapter, we surveyed background concepts and ideas used to create local feature descriptors and interest point detectors. The key concepts and ideas were also developed into the vision taxonomy suggested in Chapter 5. Distance functions were covered here, as well as useful coordinate systems. We examined the shape and pattern of local descriptors, with an emphasis on local binary descriptors such as ORB, FREAK, and BRISK to illustrate the concepts.

Feature descriptor discrimination was illustrated using image reconstructions from feature descriptor data alone. Search strategies were discussed, such as scale space pyramids and multi-level search, as well as other methods such as grid limited search. Computer vision system models were covered, including concepts such as feature space, object models, feature constraints, statistically designed features, and feature learning. Classification and training were illustrated using several methods, including kernel machines, convolutional networks, and deep learning. Several references to the literature were provided for the interested reader to dig deeper. Practical observations and considerations for designing vision systems were also provided.

In summary, this chapter provided useful background concepts to keep in mind when reading the local feature descriptor survey in Chapter 6, since the concepts discussed here were taken mainly from the current local descriptor methods in use; however, some additional observations and directions for future research were suggested in this chapter as well.